Análise de problemas de escalonamento de processos em workflow

Gregório Baggio Tramontina

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Análise de problemas de escalonamento de processos em workflow

Gregório Baggio Tramontina[1]

Março de 2004

Banca Examinadora:

- Jacques Wainer (Orientador)
- Flávio Keidi Miyazawa
  IC - UNICAMP
- Marcos Roberto da Silva Borges
  DCC - IM/UFRJ
- Arnaldo Vieira Moura (Suplente)
  IC - UNICAMP

[1]**Trabalho com suporte parcial do CNPq, processo 132918/2002-1**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Tramontina, Gregório Baggio

T684a Análise de problemas de escalonamento de processos em workflow / Gregório Baggio Tramontina -- Campinas, [S.P. :s.n.], 2004.

Orientador : Jacques Wainer.

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Fluxo de trabalho. 2. Otimização. 3. Simulação (Computadores). 4.Algoritmos genéticos I. Wainer, Jacques. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Análise de problemas de escalonamento de processos em workflow

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Gregório Baggio Tramontina e aprovada pela Banca Examinadora.

Campinas, 22 de Abril de 2004.

Jacques Wainer (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Agradecimentos

Primeiramente a Deus, por não deixar faltar força e vontade para que este trabalho se concluísse.

Ao professor orientador, Dr. Jacques Wainer, pela convivência sincera e amiga, apoio constante em todos os momentos, trabalho exímio e bom humor constante.

Aos meus amigos, peças-chave na intrincada tarefa de sobreviver e ser alguém. Aos companheiros de pós-graduação no Instituto de Computação. A todos eles um grande obrigado. Muitas páginas seriam necessárias para seus nomes, mas eles sabem quem são e estão cientes de sua importância.

A alguns amigos que estiveram mais próximos durante este tempo, inclusive ajudando na revisão de meus textos: Alexandre (Capitão), Evandro, Fernando (Moinxtro), Wesley, Raul (el mestre argentino de la matemática), Marcos Vinícius, Cléo (Billa), Marcelo "Agente Smith" Cubas, Karime, Rafael Duarte, Alan, Eduardo (Pará), Silvana, Borin, Juliana... e por aí vai.

Aos meus amigos de Bela Vista do Paraíso (Digão, Carrai, Ricardo, Manduca, Mucho Loco, Luís, Giu, Alexandro, entre tantos outros) e ao povo do Grupo Paz na Terra, por serem sempre uma outra família para mim.

À dona Cláudia, seu Carlos, e seus filhos Nicola e Gustavo, por todos os momentos juntos em sua casa, onde não só morei mas também vivi neste período em Campinas. Ao pessoal que vivia comigo nessa casa, pelos momentos divertidos e apoio a todo tempo.

Agradecimento especialíssimo à minha família: meu pai Nelson, minha mãe Alba, minha irmã Ana Regina, minha avó Avedir, e tantos outros que jamais deixaram de acreditar que este meu sonho poderia ser realizado, encorajando-me e sendo porto seguro durante as tempestades.

Ao sofrimento a mim causado, já que a dor também pode nos ensinar lições valiosas.

Ao álcool etílico, principalmente na sua forma vínica, pelas comemorações nos dias felizes e companhia nas noites tristes.

Ao Corinthians, time do coração, por ter transformado tantos dias comuns em momentos de grande comemoração pelos títulos conquistados.

A todos aqueles que não impediram a conclusão desse trabalho.

"...Busca a felicidade agora, não sabes de amanhã. Apanha um grande copo cheio de vinho, senta-te ao luar e pensa: talvez amanhã a lua me procure em vão..." (Omar Khayyam)

"A vida sem reflexão não merece ser vivida". (Sócrates)

"Tudo era um caos até que surgiu a Mente e pôs ordem no mundo". (Anaxágoras)

"O estudo sem o pensamento é inútil. O pensamento sem o estudo é perigoso". (Confúcio)

"Se pudéssemos limpar as portas da percepção, tudo se revelaria ao homem tal qual é: infinito". (William Blake)

"O hábito cria o bom e o mau, o doce e o amargo, mas, na realidade, só há átomos e vácuo". (Demócrito)

"Todas as coisas estão em eterno fluxo e mudança. Você não é, está sendo". (Heráclito de Éfeso)

"Imaginem que algumas de suas bactérias intestinais comecem a pensar e desenvolvam uma visão do mundo. Por acaso não iríamos rir delas, se declarassem que o cosmo por elas descoberto representa o mundo todo, e que as leis nele válidas valem em toda parte? Todavia, é isso que nós estamos fazendo e que fizemos durante séculos". (Mauro Maldonado, citando Paul Feyerabend, em *Não sabemos que não sabemos*, Scientific American Brasil, ano 2, núm. 21 - Fevereiro de 2004)

Resumo

A ordenação das instâncias de processos (casos) em um sistema de workflow pode trazer benefícios como a diminuição do número de casos atrasados e a minimização do tempo de processamento dos casos, entre outros. Publicações recentes em workflow reconhecem uma lacuna na pesquisa relacionada com este tema, e apontam para a literatura de escalonamento como uma possível solução. Este trabalho visa utilizar técnicas de escalonamento em um ambiente dinâmico de workflow e avaliar o desempenho dessas técnicas frente à regra FIFO (First In First Out), a política de alocação de trabalho mais utilizada nos sistemas de workflow atuais. Discute-se problemas relacionados a esta prática, e ataca-se dois deles: as incertezas quanto ao tempo de execução das atividades de workflow e as incertezas quanto às rotas que os casos seguem dentro das suas definições de processo. Para mapear essas incertezas uma nova técnica é proposta, chamada de "guess and solve", que consiste em prever os tempos de execução e rotas das atividades e resolver o problema de escalonamento determinístico resultante com uma técnica adequada, por exemplo regras de prioridade e algoritmos genéticos. Simulações cuidadosas são conduzidas e os números mostram que é quase sempre mais vantajoso utilizar outra técnica que não FIFO, e que o "guess and solve", pelo menos quando o seu erro é limitado a 30%, fornece resultados muito satisfatórios.

Abstract

Ordering cases within a worklfow can result in a significant decrease on the number of late cases and the cases' mean processing time, for example. Recent publications on workflow recognize the lack of research in this topic and points to the literature on scheduling as a possible solution. This work applies scheduling techniques to a dynamic workflow scenario and evaluates their performance in relation to the FIFO (First In First Out) rule, the most used work allocation principle in today's workflow systems. Problems related to this approach are discussed and two of them are tackled: the uncertainties regarding the activities' processing times and the cases' routes within their process definition. A new technique to map these uncertainties, called "guess and solve", is proposed. It consists of making a guess on the activities' processing times and cases' routes and then solving the resulting deterministic scheduling problem with a suitable technique, for example priority rules and genetic algorithms. Careful simulation is performed and the numbers show that it is almost always advantageous to use ordering techniques other than FIFO, and that the "guess and solve", at least when its error is bound by 30%, gives very satisfactory results.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Com as alterações de paradigmas nos sistemas produtivos e as grandes transformações ocorridas nos últimos anos nas organizações, as empresas têm adotado várias idéias para se enquadrar às novas tendências do mercado mundial [10]. Também houve uma mudança no objetivo dessas empresas em relação aos sistemas de software: enquanto antes buscava-se automatizar determinadas tarefas isoladas, agora tenta-se automatizar e gerenciar processos de forma completa [24]. Para a efetiva implementação desses novos conceitos, a tecnologia de workflow ocupa lugar de destaque nas companhias atuais.

Sistemas de workflow são usados pelas empresas para controlar e melhorar seus processos de negócio [27], onde a melhoria destes processos vem de constantemente repensá-los enquanto sua definição evolui, e da criação de melhores estratégias para escolher o potencial executor de cada atividade (por exemplo [16]). São responsáveis por instanciar, gerenciar e finalizar processos de negócio dentro do contexto de uma organização. Entre suas responsabilidades, estão aspectos de roteamento e atribuição dos itens de trabalho pendentes aos participantes do workflow responsáveis por sua execução. Durante essa atribuição, decisões quanto à ordem em que esses itens de trabalho são executados devem ser tomadas.

A maioria dos sistemas de workflow atuais fazem com que seus participantes executem os itens de trabalho na ordem em que eles chegam no sistema, ou seja, baseados em uma política FIFO (First In First Out), onde o primeiro a chegar é o primeiro a sair [28]. Outros apresentam todos os itens de trabalho ao seu potencial executor, que escolhe a seu critério um deles, o que equivale à política SIRO (Service In Random Order), onde a seleção de um elemento é feita aleatoriamente. Fica então a questão de estas serem ou não as maneiras mais recompensadoras de ordenar os itens de trabalho.

Vários trabalhos, como [15, 23, 28, 29] reconhecem a existência de aspectos de ordenação dos itens de trabalho em sistemas de workflow, e apontam para a literatura de escalonamento como uma possível maneira de se pensar no problema. Mas este mapea-

mento de um problema de workflow em um problema de escalonamento não é trivial e alguns de seus aspectos são conflitantes.

Este trabalho apresenta um estudo de problemas de escalonamento de processos em sistemas de workflow. Seu objetivo é testar o uso de técnicas de escalonamento em um cenário que possua três características consideradas importantes em um ambiente de worklflow: o cenário é dinâmico, ou seja, enquanto novos casos chegam ao sistema, outros já estão sendo processados e ainda outros estão para deixá-lo; existem incertezas quanto ao tempo de execução dos itens de trabalho em cada atividade; e existem incertezas também quanto às rotas que cada caso toma dentro da sua definição de processo. Para aferir o desempenho dessas técnicas, simulações cuidadosas do comportamento do sistema em relação a cada uma delas são realizadas. Com os resultados das simulações, busca-se responder ao questionamento levantado dois parágrafos atrás, e também levantar potenciais subsídios para a melhoria dos sistemas de workflow futuros. Pois se estes sistemas são capazes de trazer ganhos para as empresas como um todo, como mostra Rob Allen em [1], melhorá-los tem impacto diretamente positivo nos benefícios obtidos por seus usuários.

1.1 Organização do Trabalho

Este trabalho está organizado da seguinte forma: o capítulo 2 discute os conceitos básicos da tecnologia de workflow. O capítulo 3 define os principais conceitos da área de escalonamento. O capítulo 4 mostra o primeiro mapeamento realizado entre conceitos dessas duas áreas, e discute as dificuldades encontradas quando as duas se encontram. O capítulo 5 descreve como esta pesquisa estudou este encontro, mostrando a definição do problema a ser estudado, a metodologia aplicada e o desenvolvimento de suas partes técnicas. O capítulo 6 mostra os resultados obtidos, discutindo suas implicações. Finalmente, o capítulo 7 conclui o trabalho, analisando seu potencial impacto.

Capítulo 2

Workflow: Conceitos

Workflow é importante [1]. É também uma disciplina, uma prática e um conceito. Com a crescente complexidade dos processos de negócio das empresas atuais, também se torna crescente o interesse pelo seu desenvolvimento. Casos de sucesso como os comentados por Rob Allen em [1] mostram que, corretamente implementada, a tecnologia de workflow pode trazer ganhos significativos para seus usuários, que em geral são empresas imersas em um mercado cada vez mais competitivo e exigente. Este capítulo trata dos conceitos básicos de workflow, estabelecendo um conjunto de definições que servem como referência para o restante desse trabalho.

2.1 O Conceito de Workflow: Perspectiva Histórica

Delinear-se-á uma perspectiva histórica relativa ao conceito de workflow com o objetivo de mostrar o surgimento da idéia por trás das definições e termos utilizados doravante neste trabalho. Essa perspectiva histórica é retirada principalmente do trabalho de Wil van der Aalst e Kees van Hee em [28].

Os sistemas de workflow surgiram definitivamente no começo da década de 90, mas a idéia de se utilizar sistemas de informação para o suporte aos processos de negócio das empresas não é tão nova se comparada com a velocidade de desenvolvimento da computação em geral, com um passado que remonta a 1965. Nesta época, inicia-se o desenvolvimento de *aplicações descentralizadas*, cada uma com seu próprio banco de dados, definições e objetivos, para automatizar tarefas específicas e isoladas dentro das empresas. Essas aplicações executavam diretamente sobre o sistema operacional e, ou não tinham interface com o usuário, ou a tinham de maneira completamente individual. Os dados eram salvos primeiramente em cartões perfurados, e depois em fitas magnéticas, entre duas execuções consecutivas. Não havia troca de dados entre essas aplicações.

Em 1975 começa a surgir a idéia do *gerenciamento de bancos de dados*: "colocar

o gerenciamento dos dados externamente à aplicação". Esse período é marcado pelo surgimento dos *sistemas de gerenciamento de bancos de dados* (SGBD). Seu uso trazia vantagens como a centralização dos dados em um único lugar, onde todas as aplicações podem acessá-los, além de permitir que o gerenciamento desses dados seja único para todas as aplicações e que um item de dados seja gravado apenas uma vez, sem redundâncias. Os SGBD também mudaram a maneira como as aplicações eram desenvolvidas: uma vez definido o banco de dados, diferentes desenvolvedores podem trabalhar em aplicações para este mesmo banco de dados ao mesmo tempo.

O próximo gargalo no desenvolvimento de sistemas de informação estava na interface com o usuário, pois cada vez mais os softwares se tornavam interativos, e isso exigia que cada vez mais e mais tempo fosse gasto no desenvolvimento dessas interfaces. Essas eram geralmente feitas tela por tela pelos próprios desenvolvedores, e como cada desenvolvedor tem seu estilo, cada interface tinha sua forma própria de operação. Então em 1985 começam a ser desenvolvidos os *sistemas de gerenciamento de interface com o usuário* (UIMS, na sigla em inglês), que visavam resolver tais problemas "colocando as interfaces externamente às aplicações", permitindo uma definição destas de modo bem mais rápido, "convidando" o desenvolvedor a fazê-lo de uma maneira mais padronizada. Hoje as funcionalidades desses UIMSs estão embutidas em outras ferramentas, como os próprios SGBDs, ambientes de programação como o Borland Delphi e navegadores Web.

Com os dados e a interface externos às aplicações, muito do código escrito passa a ser dedicado a lidar com os procedimentos (ou processos) de trabalho das empresas. Por volta de 1995 começa a se pensar que esse componente também poderia ser isolado seguindo o exemplo dos SGBDs e UIMSs. Surge então a idéia dos *sistemas de workflow*: "colocar o gerenciamento dos processos de negócio externamente às aplicações". Assim como os bancos de dados são desenvolvidos e utilizados através de SGBDs, *sistemas de gerenciamento de workflow* (WFMS, na sigla em inglês) são utilizados para definir e gerenciar processos de negócio, instanciando-os e acompanhando sua execução até seu término. Também permitem acumular dados sobre a execução de cada processo, fornecendo um meio de rastreá-los em detalhes.

Espera-se que o uso desses sistemas continue futuramente, com o maior desenvolvimento e principalmente integração dos WFMS atuais, visto que estes ainda enfrentam muitos problemas de interoperabilidade. A *Worklflow Management Coalition* (WfMC), um consórcio de empresas provedoras de soluções de workflow, trabalha ativamente hoje na criação de padrões de desenvolvimento para esses sistemas, para possibilitar justamente tal integração.

2.2 Definições

2.2.1 Workflow

Caso procure-se uma definição do termo workflow, certamente encontraremos vários pontos de vista. Aqui reunimos alguns deles:

- a automação de um processo de negócio, durante a qual documentos, informações ou tarefas são passadas de um participante a outro, de acordo com um conjunto de regras. (WfMC) [7, 8]
- atividades que envolvem a execução coordenada de múltiplas tarefas, realizadas por diferentes entidades de processamento. (Casati et al.) [6, 25];
- atividades conjuntas que realizam interoperações com uma variedade de tarefas humanas e automatizadas. (Singh) [26];
- uma coleção de etapas de processamento (também chamadas de 'tarefas' ou 'atividades') organizadas para executar um processo de negócio. (Tatbul et al.) [11]
- uma atividade de longa duração, composta por vários passos, com algum fluxo de dados e controle entre esses passos. (Barbará et al.) [2];

Apesar de várias definições, todas são apoiadas por um tronco único: a execução de processos. Nessa execução, estão embutidas interações entre diversas partes, humanas ou automatizadas. E também um certo fluxo de informações relevantes ao contexto dos processos, e de controle. Neste trabalho adota-se na medida do possível a definição da WfMC, porém em alguns contextos o termo workflow pode ser utilizado como sinônimo de processo.

2.2.2 Atividade e Recurso

Segundo Aalst et al. [28], a atividade é uma unidade lógica de trabalho, indivisível, que é executada por um recurso. Recurso é o nome genérico dado a uma pessoa, uma máquina, ou grupos de pessoas e máquinas, que executam as tarefas. Não necessariamente o recurso é quem realmente realiza a atividade, mas sempre é o responsável por ela.

Segundo a WfMC [7, 8], a atividade é uma descrição de uma parte do trabalho que forma um passo lógico de um processo, podendo ser manual ou automatizada (de workflow). As atividades de workflow necessitam de recursos humanos ou de máquina para serem executadas, e quando é necessária a intervenção humana, a atividade é alocada para um participante do workflow. Aqui se considera a atividade como uma síntese dessas duas definições.

2.2.3 Caso, Processo e Definição de Processo

Existem muitos tipos de trabalho, tais como assar pão, arrumar a cama ou coletar resultados para uma pesquisa estatística. Em todos esses exemplos, existe algo sendo modificado ou produzido: o pão, a cama e os resultados estatísticos. Esse "algo" é aqui chamado de *caso* ou *instância*. Um caso não precisa ser algo concreto, podendo também ser um processo judicial ou uma reclamação de produto defeituoso feita ao fabricante. É diferenciável completamente de outros casos, e possui um começo e um fim bem definidos. Mas cada caso envolve um processo sendo executado, ou seja, cada caso é uma instância de execução de um processo já definido.

Um processo é um conjunto coordenado de atividades que são conectadas para se atingir um objetivo comum [7]. Para um WFMS, um processo é descrito por uma *definição de processo*, que é uma representação que suporta manipulação automatizada, e que contém critérios para seu início e término, além de informações sobre as suas atividades constituintes, como os recursos a elas associados e ferramentas necessárias para sua realização, entre outras.

Para exemplificar todos esses conceitos, tomemos a definição gráfica de um processo como o da figura 2.1. Nela encontra-se a especificação de um processo que uma empresa fictícia usa para lidar com os casos de reclamações de defeito em seus produtos, advindos de clientes. Na figura, os retângulos com laterais arredondadas representam as atividades do processo; os triângulos marcados com a letra "o" representam pontos onde há uma separação de caminhos: o caso deve seguir apenas uma das rotas que saem do triângulo; também representam o fechamento dessa diferenciação de caminhos, ou seja, o ponto onde a diferenciação termina; retângulos pequenos marcados com a letra "e" representam tanto a abertura quanto o fechamento de atividades paralelas, que devem ser executadas ao mesmo tempo; por fim, losangos marcados com a letra "l" representam iterações, onde há um ponto de retorno a uma atividade anterior, o que configura a formação de um laço ou *loop*.

Muitas ferramentas de workflow permitem que se defina processos graficamente como o da figura 2.1, traduzindo-os depois para outros formatos digitais aceitos pelo WFMS como uma descrição de processos.

2.2.4 Itens de Trabalho

Segundo Aalst et al. [28], um item de trabalho (*work item*, em inglês), é a junção de uma instância do processo (caso) com uma atividade pela qual esta instância deve passar. A execução de uma atividade é na verdade a execução de um item de trabalho. Ou seja, a atividade é a definição do que fazer em um determinado passo do processo, mas é genérica. Quando há um caso que requer que determinada atividade seja executada, então para este

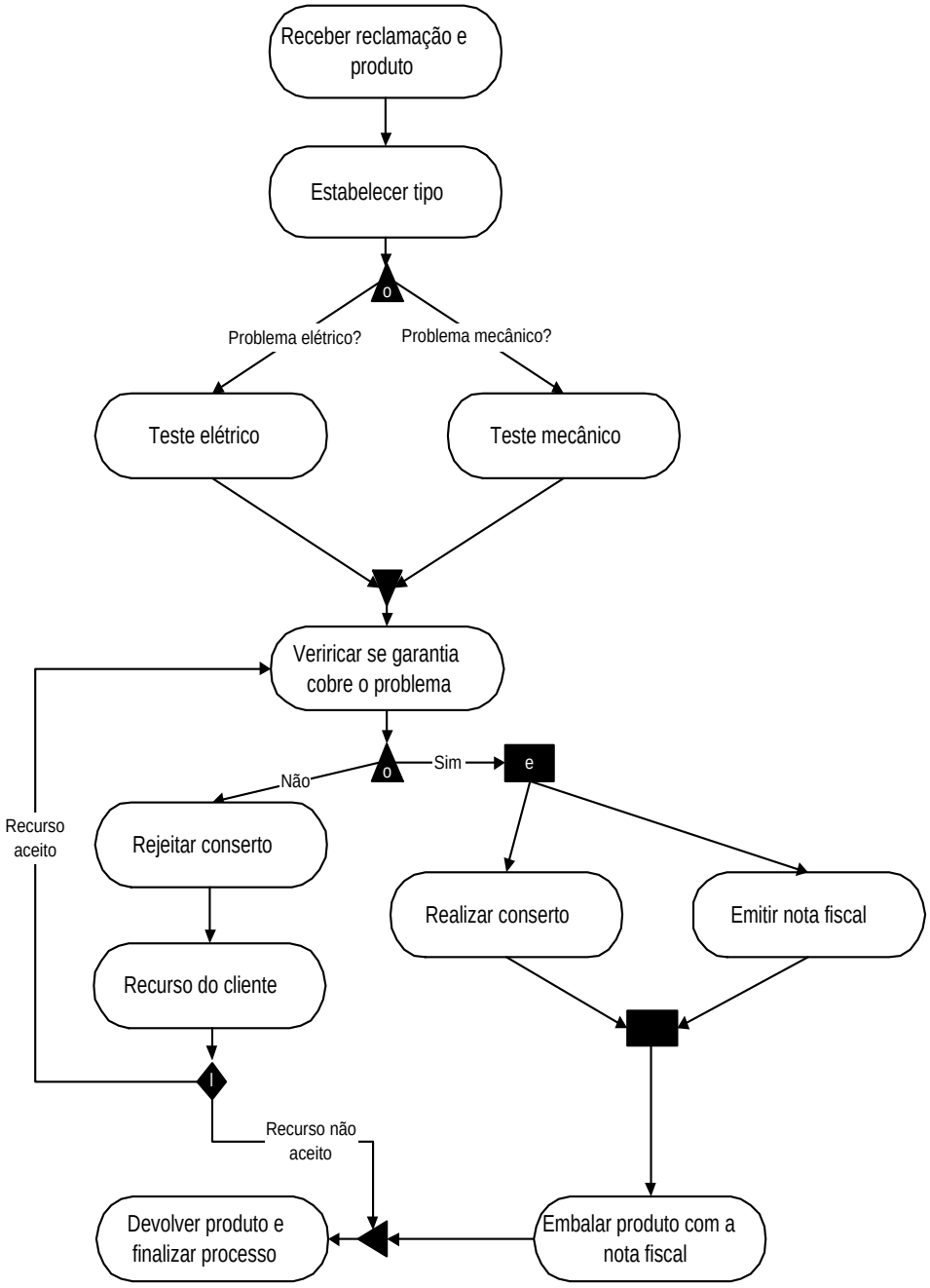


Figura 2.1: Exemplo de definição de processo

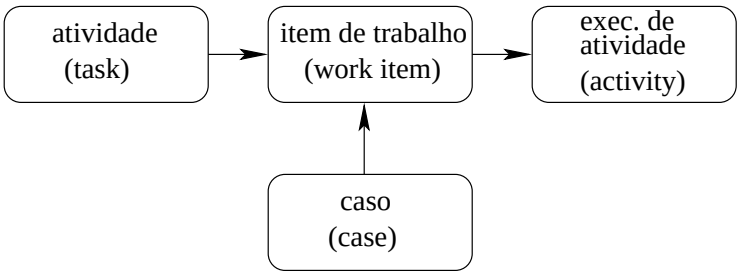


Figura 2.2: Relação entre os termos atividade, item de trabalho, execução de atividade e caso

caso é criado um item de trabalho relativo à atividade em questão, e que vai ser executado por um recurso específico; quando o item de trabalho está sendo efetivamente executado, então se fala em execução da atividade.

Na nomencaltura em inglês, os termos utilizados são ligeiramente diferentes no que diz respeito às atividades e sua execução. O termo *task* é utilizado para designar uma atividade, enquanto *activity* é usado para designar a execução da atividade. A relação dos termos atividade, caso, item de trabalho e execução de atividade pode ser vista na figura 2.2.

2.3 Sistemas de Gerenciamento de Workflow

Embora um sistema de workflow possa ser organizado de forma manual, o que se busca é sempre a utilização de ferramentas computacionais que forneçam suporte automatizado à realização do trabalho. Aqui se colocam os sistemas de gerenciamento de workflow ou simplesmente WFMS.

Um sistema de gerenciamento de workflow é um sistema que completamente define, gerencia e executa workflows, através de software cuja ordem de execução é dada por uma representação computacional da lógica dos workflows, ou seja, um sistema que é capaz de interpretar as definições de processos, interagir com os participantes dos workflows e, quando necessário, invocar as ferramentas (computacionais) necessárias para a realização de uma atividade [7, 8].

Apesar de existirem no mercado vários produtos diferentes que se colocam como ferramentas de workflow, existem características comuns entre eles no que diz respeito às suas funções principais. A WfMC [8] identifica que, em um nível mais alto, todos os WFMS têm as seguintes funcionalidades:

- funções de "tempo de construção" (*build-time functions*), que lidam com a definição

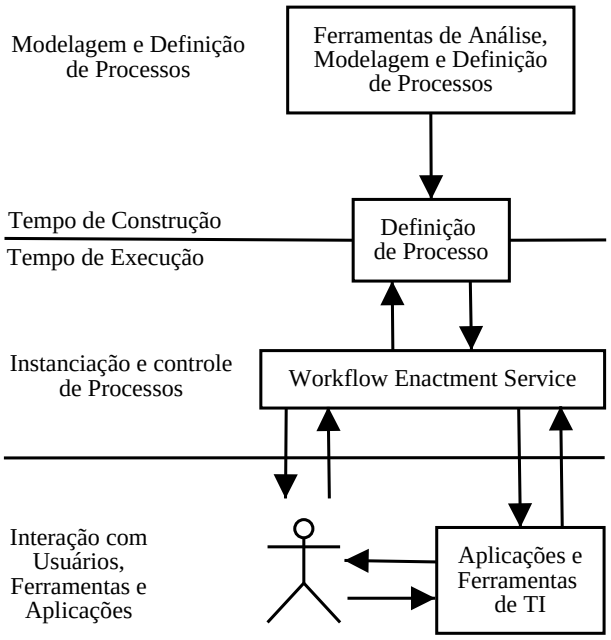


Figura 2.3: Relação entre as principais funções de um WFMS

e modelagem dos processos de workflow e suas atividades constituintes; o resultado final dessa fase é a definição de processo acabada;

- funções de "tempo de execução" (*run-time functions*), que gerenciam e executam os processos de workflow em um ambiente operacional, além de seqüenciar e alocar as atividades necessárias para a execução desses processos; são essas funções que cuidam de criar uma instância de processo para lidar com um caso específico, escalonar suas atividades para os recursos, e acompanhar esse caso até o término de sua execução;

- funções de interação, em tempo de execução, com os usuários e ferramentas computacionais para o processamento dos vários passos de cada atividade;

A relação entre esses itens pode ser vista na figura 2.3. As setas indicam o sentido das interações.

2.3.1 Workflow Engine e Workflow Enactment Service

Segundo a WfMC, uma *workflow engine* é uma parte de um WFMS que provê o ambiente de execução para as instâncias de processo [7]. É a parte responsável por criar, gerenciar

e executar tais instâncias.

Uma ou mais workflow engines formam um domínio de workflow, ou seja, um ambiente homogêneo de execução dos processos. Esse domínio é gerenciado pelo *workflow enactment service*, ou WES, que provê o suporte necessário para a execução das instâncias de processo pelo conjunto de workflow engines que o compõem [7, 8]. Assim, um WES é formado por uma ou mais workflow engines.

2.3.2 Roteamento e Alocação de Itens de Trabalho

Uma das principais funções de um WFMS é o roteamento e a alocação dos itens de trabalho dentro do escopo dos processos por ele gerenciados. O roteamento diz respeito ao caminho particular que cada caso segue dentro de uma definição de processos, e que só é conhecido durante a execução deste. A alocação é o processo de designar esses itens de trabalho aos recursos que podem realizá-lo.

Para o roteamento, existem quatro situações possíveis com as quais o WFMS deve lidar [28]:

- roteamento *seqüencial*: a mais simples; ocorre quando duas ou mais atividades devem ser executadas em seqüência;
- roteamento *paralelo* (AND): quando duas atividades podem ser executadas independentemente e ao mesmo tempo, sem que uma influencie na outra; são disparadas em determinado ponto do processo e ressincronizadas em um ponto mais adiante;
- roteamento *condicional ou seletivo* (OR): acontece quando a próxima (ou próximas) atividade a ser executada depende de uma escolha, que por sua vez depende das propriedades específicas de cada caso; o resultado dessa escolha determina qual será essa próxima atividade;
- roteamento *iterativo*: representa atividades que devem ser executadas em *loop*, até que determinada circunstância seja atingida; um bom exemplo são atividades que dependem de uma validação final: caso o resultado dessa validação seja negativo, as atividades devem ser repetidas;

Quanto à alocação dos itens de trabalho, é função do WFMS designá-los a um determinado recurso para que estes sejam levados adiante. Essa atribuição de trabalho deve obedecer restrições como não alocar serviço a alguém não autorizado, e quando existe mais de um item de trabalho para um recurso, o sistema pode atuar sobre a ordem de execução destes. Nos sistemas de workflow atuais geralmente se utiliza a política FIFO (*Firts In First Out*) para designar os itens de trabalho aos recursos, ou seja, o sistema

faz com que tais itens de trabalho sejam executados em ordem de sua chegada. Outra política utilizada é a de deixar que o próprio recurso escolha qual item de trabalho ele irá executar, dentro daqueles disponíveis a ele. Como veremos adiante, essa ordenação é o centro da pesquisa descrita neste trabalho.

2.4 O Modelo de Referência da WfMC

Muitas ferramentas de workflow surgiram desde que essa tecnologia começou a tomar forma. Mas a preocupação com a integração entre essas ferramentas não existia, o que levou ao desenvolvimento de várias soluções ilhadas entre si. Com o obejtivo de promover padrões para o desenvolvimento de ferramentas de workflow, várias empresas desenvolvedoras dessa tecnologia se juntaram para criar a *Workflow Management Coalition*, ou WfMC. Esta propôs em 1995, em [8], um modelo de referência para sistemas de gerenciamento de workflow. O principal objetivo desse modelo é promover a interoperabilidade entre várias ferramentas de workflow de desenvolvedores diferentes, mas ele também fornece uma visão ampla das funcionalidades dos sistemas de workflow como um todo.

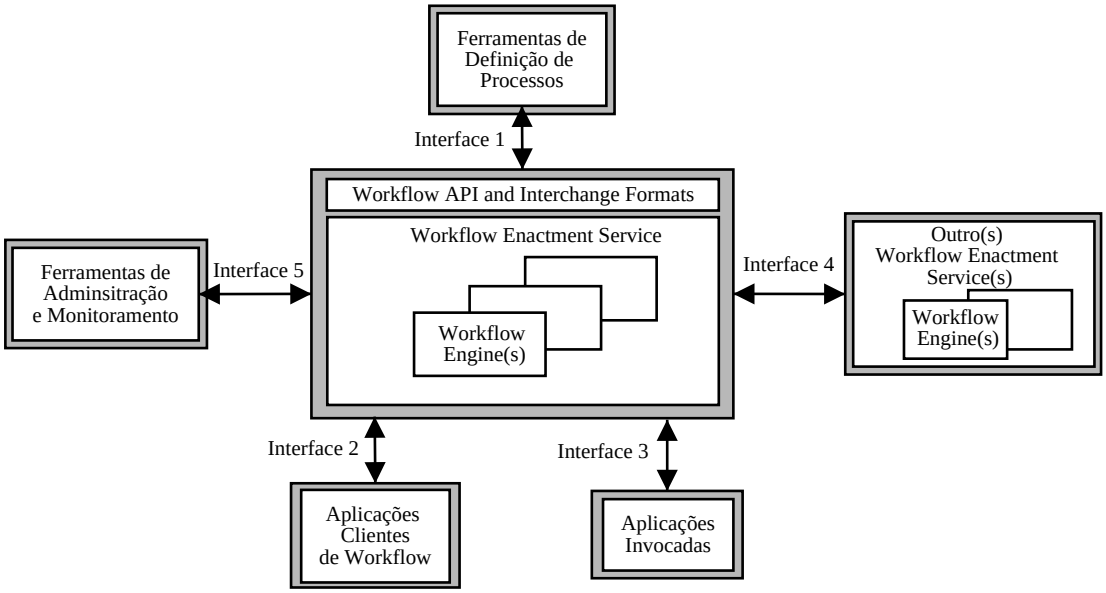


Figura 2.4: Modelo de referência para sistemas de workflow da WFMC

A figura 2.4 demonstra o modelo da WfMC. Pode-se notar que o centro do modelo é tomado pelas workflow engines, que ficam dentro do workflow enactment service (WES). Este último é encapsulado pela Workflow Applications Programming Interface (WAPI),

que possui cinco interfaces numeradas de 1 a 5, cada uma cuidando da interação do WES com outros componentes do modelo. A WAPI representa a proposta da WfMC de um meio para a interoperação entre as ferramentas de workflow, e consiste em um conjunto de funções através das quais os componentes do modelo interagem uns com os outros. Isso visa permitir que cada uma das partes do modelo seja projetada por um desenvolvedor diferente, mas que todas possam agir juntas em um único WFMS, através da WAPI.

As interfaces são as principais partes da WAPI. Cada uma define um bloco lógico de interação entre as diversas partes de um sistema de workflow. A interface 1 cuida da interação entre as ferramentas de definição de processos e o WES. A interface 2 permite o acesso de aplicações clientes às listas de itens de trabalho dos participantes dos workflows; essas aplicações clientes são consideradas como aquelas que realizam a junção entre participante do workflow e itens de trabalho a ele pendentes. A relação entre o WES e as ferramentas computacionais por ele invocadas é definida na interface 3, pela qual pode-se interagir com as aplicações e passar algum item de trabalho para elas, e depois receber os resultados de volta. Na interface 4 define-se a interação de um WES com outros externos a este, para a possível troca e/ou execução conjunta de processos comuns. Por último, na interface 5, explicita-se a comunicação entre ferramentas de administração e monitoramento do sistema com o WES. A WfMC continua a trabalhar nessas interfaces, não estando todas ainda completamente definidas.

Capítulo 3

Escalonamento: Conceitos

Escalonamento é um processo que existe na maioria dos sistemas de produção e manufatura, bem como na maioria dos sistemas de processamento de informações. Ele também aparece em problemas de transporte e distribuição e em outros tipos de indústrias de prestação de serviços [21]. Este capítulo trata dos conceitos fundamentais da teoria de escalonamento que, como no capítulo anterior, estabelecem a base teórica para o restante deste trabalho.

3.1 Definição

Segundo Pinedo [21], escalonamento é um processo de tomada de decisões que se preocupa com a alocação de recursos limitados para tarefas ao longo do tempo, e possui como meta a otimização de uma ou mais funções-objetivo. Tais recursos e tarefas podem tomar muitas formas. Os recursos podem ser máquinas em um chão de fábrica, pistas em um aeroporto, processadores em um sistema de computação, entre outros. As tarefas podem ser operações em um processo de produção, pousos e decolagens em pistas de um aeroporto, execução de software em um sistema de computação, entre outros. As funções-objetivo também podem ser diversas, como a minimização do tempo médio gasto por uma atividade de montagem de peças em uma máquina de uma linha de produção ou a minimização do número de pousos e decolagens atrasadas em um aeroporto. Na literatura científica, uma tarefa é chamada de *job* e um recurso, de *máquina*.

3.2 Notação para Problemas de Escalonamento

Durante as últimas décadas um número considerável de pesquisas teóricas no campo de escalonamento foi desenvolvido [21]. Nesse tempo, uma notação específica para esse

tipo de problema tomou forma. Esta notação, introduzida por Graham et al. em [13], é também chamada de *notação de Graham*. Nesta seção discute-se tal notação, o que dará a oportunidade de apresentar os tipos mais conhecidos de problemas de escalonamento, conjuntamente com suas noções teóricas. A base para essa apresentação pode ser encontrada em [20, 21].

O número de *jobs* a serem executados é aqui considerado finito, e representado por n. O número de máquinas que se possui para executá-los é denotado por m. Geralmente, o índice j refere-se a um job, e o índice i, a uma máquina. Se o job j requer processamento, então o par (i, j) refere-se ao processamento do job j na máquina i. As informações seguintes estão associadas ao job j.

- **processing time ($p_{i,j}$):** $p_{i,j}$ representa o tempo gasto para que o job j seja processado na máquina i; se esse tempo não depende da máquina onde j é executado, então omite-se o índice i, ficando apenas p_j;
- **release date (r_j):** também conhecida como *ready date*, representa o tempo de chegada do job ao sistema, ou seja, o job só pode começar a ser executado a partir desse instante de tempo;
- **due date (d_j):** o prazo para o qual o job deve estar pronto (para quando o job foi prometido ao cliente);
- **weight (w_j):** o peso do job j; pode ser entendido como uma medida de importância dada ao job;

Segundo a notação de Graham, um problema de escalonamento é descrito por uma tripla, composta dos termos α, β, e γ, e denotada por $\alpha|\beta|\gamma$. O termo α descreve o ambiente das máquinas que processam os jobs, e possui um único valor. O termo β descreve características de processamento dos jobs e pode conter nenhum, apenas um, ou múltiplos valores. Por último, γ denota a função-objetivo a ser minimizada e geralmente contém apenas um valor.

3.2.1 Valores de α

Esta seção descreve possíveis valores do termo α, conjuntamente com a descrição dos tipos de problemas que denotam.

- **single machine (1):** é o caso onde apenas se tem uma máquina ou processador; é um caso especial de todos os outros ambientes mais complexos;

- **identical machines in parallel (Pm):** denota que existem m máquinas idênticas em paralelo; o job j requer uma operação apenas, e pode ser processado em qualquer uma das máquinas;

- **machines in parallel with different speeds (Qm):** neste caso, existem m máquinas em paralelo, mas estas diferem na velocidade que podem executar os jobs; a velocidade de uma máquina i é denotada por v_i; a velocidade que o job j gasta para ser processado na máquina i ($p_{i,j}$) é p_j/v_i; este ambiente também é referenciado com o nome de *máquinas uniformes*.

- **unrelated machines in parallel (Rm):** uma generalização do ambiente anterior; existem m máquinas diferentes em paralelo; a máquina i pode processar o job j com velocidade v_{ij}; o tempo de processamento $p_{i,j}$ do job j na máquina i é dado por $p_j/v_{i,j}$; pode-se notar que agora a velocidade da máquina depende do job a ser processado nela.

- **flow shop (Fm):** existem m máquinas em série; cada job deve ser processado em todas elas; todos os jobs têm a mesma "rota", ou seja, devem todos serem processados primeiro na máquina 1, depois na 2, e assim por diante; depois de terminar sua execução em uma máquina, o job vai para a fila de execução da próxima máquina; geralmente essas filas operam sob a lógica FIFO (first in first out); se isto for verdade, então esse flow shop pode ser chamado de um *permutation flow shop* e o valor *prmu* deve ser incluído no campo β.

- **flexibe flow shop (FFs):** é uma generalização do flow shop e do ambiente com máquinas paralelas; agora existem s estágios em série, e em cada um deles há um número de máquinas em paralelo; cada job deve ser processado primeiro no estágio 1, depois no 2, e assim por diante; geralmente, cada job requer apenas uma máquina, e pode ser processado por qualquer uma delas, dentro de um estágio;

- **open shop (Om):** aqui, cada job deve ser processado novamente em cada uma das m máquinas; no entanto, alguns desses tempos de execução podem ser zero; não há restrições quanto ao percurso dos jobs no ambiente das máquinas; o escalonador pode tomar decisões quanto a esse aspecto;

- **job shop (Jm):** em um job shop, cada job tem sua própria rota para seguir; distinções são feitas nos casos onde cada job deve "visitar" cada máquina ao menos uma vez, e quando é permitido aos jobs passarem por uma mesma máquina mais de uma vez; nesse último caso, o campo β deve conter a entrada *recrc*, que significa *recirculation*;

3.2.2 Valores de β

O campo β da definição de problemas de escalonamento indica condições especiais para a execução dos jobs. Mostramos aqui possíveis valores desse termo, bem como a especificação de seu significado.

- **release dates (r_j):** quando os jobs têm uma data mínima para que sua execução seja iniciada, o símbolo r_j deve aparecer no campo β; ou seja, isso indica que o job j não pode começar seu processamento antes de sua *release date* r_j; caso este símbolo não esteja presente, os jobs podem ser processados a qualquer momento; note que caso os jobs tenham *due dates* (prazos), nada é especificado aqui; a função objetivo, do campo γ, será suficiente para que saibamos se esse aspecto é relevante ao problema;

- **sequence dependent setup times (s_{jk}):** o símbolo s_{jk} indica que o tempo de setup[1] dos jobs é dependente da seqüência entre eles; ou seja, haverá diferentes setup times para um job k dependendo de qual job j esteja à sua frente na execução;

- **preemptions (*prmp*):** *preemption* diz respeito à possibilidade de se interromper a execução de um job antes dela ter sido terminada, e depois poder retornar a ela; o tempo já processado do job não é perdido; quando ele volta a ser executado, apenas permanece na máquina o tempo que lhe falta para ser concluído; quando o valor *prmp* está presente no campo β, ela é permitida; caso contrário, não se pode ter *preemption*;

- **precedence constraints (*prec*):** indica que existe uma relação de precedência entre os jobs, de forma que um ou mais jobs devem ser executados antes que outros o sejam;

- **breakdowns (*brkdwn*):** implica que as máquinas não estão continuamente disponíveis; o tempo que uma máquina fica indisponível pode ser considerado fixo, ou ser modelado através de distribuições de probabilidade;

- **machine eligibility restrictions (M_j):** pode figurar como valor para o campo β quando em α se tem o valor Pm, ou seja, o problema lida com m máquinas em paralelo; quando existe essa restrição, nem todas as máquinas são capazes de executar o job j, mas apenas aquelas pertencentes ao conjunto M_j;

[1] O tempo gasto para preparar a execução de um determinado job, como por exemplo o tempo que se leva para colocar todos os ingredientes de uma vitamina no liquidificador, antes de propriamente batê-la.

- **permutation (*prmu*):** pode aparecer em um flow shop (Fm) quando as filas entre cada máquina operam de acordo com a política FIFO; isso implica que a ordem em que os jobs são executados na primeira máquina é mantida durante toda a execução;
- **blocking (*block*):** pode acontecer em um flow shop quando as filas de espera de execução entre as máquinas possuem tamanho limitado; deste modo, se um job j termina seu processamento em uma máquina i, mas a fila (*buffer*) da máquina $i+1$ está cheia, então j não pode sair de i, o que faz com que os outros jobs que esperam por execução em i sejam bloqueados; *block* implica em *prmu*;
- **no-wait (*nwt*):** também pode ocorrer em flow shops; essa condição quer dizer que os jobs não podem esperar entre duas máquinas; isso implica que, caso necessário, o momento da execução de um job na primeira máquina deve ser adiado para garantir que ele possa passar por todas as outras sem esperar;
- **recirculation (*recrc*):** pode ocorrer em um job shop, no caso dos jobs poderem passar pela mesma máquina mais de uma vez;

3.2.3 Valores de γ

Em γ é especificada a função que se deseja minimizar no problema dado. Geralmente são funções dos *completion times* dos jobs, que por sua vez dependem da ordem de execução destes. Também podem estar envolvidas funções que dependem das *due dates*, ou os prazos, dos jobs. Para precisarmos as funções-objetivo mais comuns, faz-se necessário algumas definições prévias.

O *completion time* de um job j em uma máquina i é denotado por C_{ij}. O completion time desse job na última máquina onde ele requer processamento, ou seja, o instante em que ele sai do sistema, é representado por C_j. Como já mencionado, a due date desse job é representada por d_j.

Define-se *lateness* de um job j como sendo

$$L_j = C_j - d_j$$

que é positiva quando j está atrasado, e negativa quando está adiantado. Também se define *tardiness* de um job j como

$$T_j = \max(C_j - d_j, 0) = \max(L_j, 0) \ . \tag{3.1}$$

A diferença entre lateness e tardiness é que o tardiness nunca é negativo. Por isso, não se preocupa se o job está adiantado, apenas em quanto está atrasado. Já o lateness fornece também uma idéia de quanto tempo o job está à frente de seu prazo de término,

visto que seu valor é negativo quando o job está adiantado, e pode ser útil para análises que requerem esse tipo de informação.

Também se define a *unit penalty* de um job j como

$$U_j = \begin{cases} 1 & \text{se } C_j > d_j \\ 0 & \text{caso contrário} \end{cases} \quad . \tag{3.2}$$

onde se pode notar que U_j vale 1 quando o job j está atrasado e 0 caso contrário. Agora podemos apresentar possíveis valores do campo γ.

- **makespan (C_{max}):** o makespan é equivalente ao completion time do último job a deixar o sistema; é definido como

  $$C_{max} = \max(C_1, C_2, \ldots, C_n) \ .$$

  Geralmente um makespan baixo indica uma alta taxa de utilização das máquinas;

- **maximum lateness (L_{max}):** definida como

  $$L_{max} = \max(L_1, L_2, \ldots, L_n)$$

  mede a pior violação de prazos do sistema;

- **total weighted completion time ($\sum w_j C_j$):** dá uma idéia do custo total de estocagem que a ordenação utilizada traz; essa soma é freqüentemente referenciada na literatura como *weighted flow time*.

- **discounted total weighted completion time ($\sum w_j(1 - e^{-rC_j})$):** é uma media mais geral do que a anterior; aqui, o custo é descontado de acordo com um fator r para cada unidade de tempo, onde $0 < r < 1$;

- **total weghted tardiness ($\sum w_j T_j$):** a soma ponderada da função tardiness de cada job;

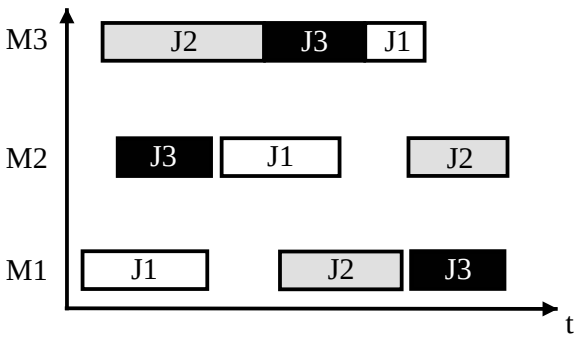
- **weighted number of tardy jobs ($\sum w_j U_j$):** o número de jobs que estão atrasados; como pode ser medido com facilidade, é um importante critério inclusive em aplicações reais;

De acordo com esta notação, $F3|prmu|\sum w_j C_j$ denota um *flow shop* com 3 máquinas, ou seja, as 3 máquinas estão em série; as filas entre as máquinas operam sobre a política FIFO (*prmu*), e o objetivo é minimizar a soma ponderada dos *completion times* dos jobs. Já $1|s_{jk}|C_{max}$ define que existe apenas 1 unidade de processamento, onde os jobs possuem setup times que dependem da seqüência de sua execução, e o objetivo é minimizar o *makespan*. Sabe-se que esse último é equivalente a resolver o problema do caixeiro viajante, onde o caixeiro deve visitar n cidades de forma que a distância total viajada deve ser minimizada.

3.2.4 Resolução de um Problema de Escalonamento

Definido o problema, o objetivo torna-se então gerar uma atribuição da execução dos jobs para as máquinas, durante um período de tempo, que otimize a função-objetivo desejada. Esta atribuição é chamada de um *escalonamento*, ou *schedule* no termo original. O termo "escalonamento" pode gerar alguma confusão em Português, já que na literatura, o termo *scheduling*, que quer dizer a operação de gerar um *schedule* também se traduziria como escalonamento aqui. Portanto, neste trabalho, o termo original *schedule* é utilizado.

Existem algumas formas de se representar um schedule. Uma delas é a ilustrada na figura 3.1-(a), que é um diagrama que indica a ocupação das máquinas, pelos jobs, durante o tempo. A outra é uma tabela contendo os tempos de início do processamento dos jobs em cada máquina, ilustrada na figura 3.1-(b). Na figura 3.1, as máquinas são representadas por M1, M2 e M3, e os jobs por J1, J2 e J3. Para este trabalho, a segunda forma de representação será mais útil, e servirá de base para a implementação da simulação do problema de escalonamento estudado.



	J1	J2	J3
M1	2	32	49
M2	20	48	10
M3	41	6	34

(b)

Figura 3.1: Formas de se representar um schedule

3.3 Outros Conceitos

3.3.1 Escalonamento Estático e Dinâmico

A diferença entre uma operação de escalonamento estático e dinâmico é que, no primeiro, todos os jobs são conhecidos de antemão. O objetivo é escalonar estes jobs apenas uma vez, sendo este escalonamento fixo, já que não é necessário mudá-lo. Já em um escalonamento dinâmico, o sistema não possui apenas um conjunto de jobs fixo, mas estes chegam ao sistema em pontos assíncronos de tempo, e enquanto novos jobs chegam, outros estão

executando. A cada chegada ou saída de um job no/do sistema, o conjunto de jobs conhecidos muda.

A simples inserção de um novo job no sistema sem uma revisão do schedule já gerado pode levar a uma deterioração dos resultados deste último, e por isso tal dinamismo leva a questões como o reescalonamento (*rescheduling*), onde a preocupação não é gerar um schedule novo a partir do zero, mas alterar o já existente de forma a contemplar o novo job, tentando ser o menos complexo possível nesse procedimento.

3.3.2 Escalonamento Determinístico e Estocástico

A literatura em escalonamento trata de problemas determinísticos. Tais problemas são aqueles onde os valores de suas variáveis, como tempos de execução dos jobs em cada máquina, release dates e due dates, entre outros, são previamente conhecidos e exatos. Mas muitas vezes as informações que temos para ponderar sobre escalonamento são poucas, e sujeitas a variações dos mais diversos tipos. Por exemplo, na maioria das situações reais não se pode prever, ao menos com exatidão, quando alguma máquina vai ter problemas e acarretar perdas no processo. Ou então os dados que a companhia possui sobre sua própria produção são insuficientes para traçar panoramas determinísticos de modelagem. Desse modo, há também uma vertente de escalonamento que lida com essas incertezas de maneira probabilística, o que Pinedo [21] chama de *stochastic models*, ou modelos estocásticos, de escalonamento.

Nos modelos estocásticos, não se sabe os valores exatos dos tempos de processamento, *release dates* e prazos dos jobs, mas sim suas distribuições de probabilidade. Seus reais valores só serão conhecidos depois do processamento ter sido realizado. Toda a análise é feita raciocinando-se sobre probabilidades. A notação utilizada para estes casos não é totalmente compatível com a anteriormente apresentada. Existem novos símbolos e conceitos, como *médias* (valores esperados) para as variáveis. Faz-se uso de funções de densidade e probabilidade, e dominância estocástica, entre outros conceitos . Para uma detalhada descrição de tais problemas, consulte [21].

3.3.3 Tipos de schedules

Existem tipos de schedules, dependendo de algumas de suas características. Aqui apresenta-se os três mais relevantes a este trabalho, como estão definidos por Pinedo em [21].

- **Non-delay**: um schedule onde nenhuma máquina é mantida parada enquanto existem operações disponíveis para processamento.

- **Active**: um schedule onde nenhuma operação pode ser completada mais cedo através da alteração da ordem de processamento nas máquinas, sem que essa alteração atrase outras operações.
- **Semi-active**: um schedule onde nenhuma operação pode ser completada mais cedo sem alterar a ordem de processamento nas máquinas.

Um schedule *active* é *semi-active*, mas o contrário não é necessariamente verdadeiro.

Capítulo 4

Workflow e Escalonamento

Muitas publicações, como as apresentadas em [15, 23, 28, 29], reconhecem a existência de aspectos de escalonamento em sistemas de worflow. Particularmente, Aalst e Hee [28] definem muito bem o ponto onde essas áreas se encontram.

> *"Se existe um excesso de trabalho em determinados instantes de tempo, não podemos transformar cada [item de trabalho] em uma atividade imediatamente. Podem existir mais itens de trabalho do que recursos disponíveis. Se este for o caso, então uma escolha deve ser feita quanto à ordem na qual esses itens de trabalho são selecionados."*

No entanto, a utilização de técnicas de escalonamento em workflow não é trivial. Neste capítulo expõe-se os primeiros passos da pesquisa desenvolvida, mostrando o mapeamento realizado de conceitos básicos de workflow em conceitos de escalonamento, e discutindo os problemas encontrados quando essas duas áreas se encontram.

4.1 Mapeamento Workflow-Escalonamento

Muitos conceitos de workflow e escalonamento compartilham das mesmas idéias básicas [27]. Aalst e Hee [28] colocam que o roteamento de um caso (de workflow) através de vários recursos exibe muitas similaridades com o roteamento de um produto através das máquinas de um departamento de produção, onde as técnicas de escalonamento são ferramentas presentes. Então, o primeiro passo da pesquisa consistiu em identificar estas similaridades iniciais, traduzindo-as em um mapeamento de conceitos básicos de workflow em conceitos de escalonamento. A tabela 4.1 demonstra esse mapeamento.

Pela tabela 4.1, pode-se notar que uma instância de processo sendo executada pelo WFMS é considerada um job em um sistema de escalonamento. A execução de uma

Workflow	Escalonamento
instância de processo (caso)	job
atividade	passo de execução de um job
aplicações/pessoas	máquinas

Tabela 4.1: Mapeamento de conceitos básicos de workflow em escalonamento

atividade é vista como um passo de execução de um job em uma máquina. Um recurso de workflow (aplicações/pessoas e máquinas) é mapeado como uma máquina do ambiente de escalonamento. Esse mapeamento permitiu a aplicação das técnicas de escalonamento às partes corretas do sistema de workflow estudado, guiando os esforços de pesquisa subseqüentes.

4.2 Características da Junção de Workflow e Escalonamento

O caso geral de um problema de escalonamento é NP-difícil [3], e portanto avanços na área se fazem principalmente através da exploração de especificidades do problema. Por exemplo, em um *flow shop*, todos os jobs devem seguir uma mesma seqüência de passos de execução (ou rota), como se as máquinas desse problema estivessem em série. Já em um *job shop*, cada job tem uma rota própria para seguir.

Em um primeiro momento, a ordenação dos casos em workflow parece estar entre um *flow shop* e um *job shop*: se existe apenas um processo no sistema de workflow, todos os casos vão seguir a rota demarcada por este processo; se existem vários processos, então o cenário se parece mais com um *job shop* organizado em blocos, ou seja, existirão subconjuntos do conjunto de todos os casos no sistema, onde os casos de um mesmo subconjunto seguem a mesma rota, mas casos de subconjuntos diferentes seguem rotas diferentes. Tais abordagens são incorretas pois quando workflow e escalonamento se encontram, muitos problemas surgem, derivados das características que o cenário de workflow empresta ao de escalonamento. Os principais desses problemas foram levantados como parte da pesquisa em [27]. Esta seção faz uma revisão do que já foi publicado no artigo mencionado.

4.2.1 Incertezas Quanto ao Tempo de Execução das Tarefas

Sistemas de workflow lidam com casos de trabalho reais, geralmente envolvendo processos de nível mais alto que processos de linhas de montagem ou chão de fábrica. Enquanto

que nesses últimos é possível saber com boa precisão quanto tempo uma atividade leva para ser executada, como um robô que solda as partes metálicas de um carro, por exemplo, nos primeiros é mais difícil fazê-lo devido à intensa ligação destas com fatores não controláveis, principalmente o fator humano. Em um pequeno exemplo, um banco pode demorar mais tempo para aprovar um empréstimo a um determinado cliente que precise levantar mais documentação do que a outro que já a possui de antemão. Isso conduz ao não conhecimento prévio de quanto tempo cada atividade leva para ser executada por determinado recurso, embora estimativas baseadas em conhecimento acumulado de casos passados, ou especialistas com muitos anos de experiência e observação prática, sejam possíveis.

Em escalonamento, a maioria das pesquisas é focada em ambientes onde o tempo de execução das tarefas é exato e conhecido previamente. Quando não se tem certeza desses valores, o problema se torna estocástico. Para transformar um problema de natureza estocástica em um problema determinístico pode-se fazer com que os tempos de execução assumam valores médios esperados, juntamente ou não com um fator de perturbação desses tempos esperados, como fazem Lawrence e Sewell em [18]. Sendo esta uma das características levadas em consideração durante a pesquisa, utilizou-se uma abordagem ligeiramente diferente da apontada pelos autores citados, onde os tempos de execução de cada atividade assumem valores oriundos de uma previsão, com erro controlado, sobre seus valores reais.

4.2.2 Roteamento Paralelo

O roteamento paralelo em workflow faz com que um mesmo caso tenha duas ou mais atividades sendo executadas ao mesmo tempo. Em contrapartida, uma situação onde um job executa em mais de uma máquina no mesmo instante não foi um assunto encontrado na literatura de escalonamento durante a pesquisa. Uma possível solução é o mapeamento caso-job ser feito de maneira diferente, onde um caso pode ser mapeado em um número variável de jobs durante seu ciclo de vida. Não se conhece o impacto disso no foco da pesquisa.

4.2.3 Incertezas Quanto às Rotas dos Casos

Em uma definição de processo como a apresentada na figura 2.1 estão definidos os possíveis caminhos que uma instância desse determinado processo pode percorrer. Pode-se notar que, quando da primeira decisão a ser tomada, o caso pode assumir dois caminhos diferentes: seguir para um teste elétrico ou mecânico. Pode não ser possível previamente determinar qual destes dois caminhos o caso irá seguir. Isso claramente gera uma incerteza quanto à rota que tal caso fará dentro da definição de seu processo, incerteza

essa ligada diretamente à estrutura de roteamento condicional (*OR-split*) já comentada na seção 2.3.2.

Em escalonamento, durante a pesquisa, não foram encontrados exemplos de modelos que lidam com esse tipo de incerteza. Os jobs sempre têm caminhos bem conhecidos para seguir. Como esta é uma característica atacada por nossa pesquisa, abordá-la diferenciadamente da teoria clássica de escalonamento também foi necessário.

4.2.4 A Escolha da Função-Objetivo

O *flowtime*, ou a soma dos *completion times*, dos jobs é a função-objetivo mais analisada pela literatura de escalonamento, ou seja, a maioria das pesquisas tenta fazer com que os jobs sejam completados o mais rápido possível. Embora essa também seja uma das funções de um sistema de worklfow, objetivos relacionados com o atraso dos casos são igualmente relevantes, tais como a porcentagem de casos atrasados e o atraso médio por caso atrasado no sistema. Além disso, pode haver casos onde uma combinação de funções-objetivo seja mais útil. Pinedo, em [21], demonstra uma maneira de combinar diferentes regras de prioridade para tentar abranger diferentes funções-objetivo, mas uma outra abordagem pode ser necessária para sistemas de workflow.

4.2.5 Atividades *Preemptive* e não-*Preemptive*

Em workflow muitas das atividades são inerentemente *preemptive*. Como exemplo, podemos citar a escrita dessa dissertação, que foi pausada e reiniciada diversas vezes. Outras atividades são inerentemente não-*preemptive*, como participar de uma reunião de negócios para a tomada de decisões estratégicas urgentes de uma empresa. Os dois tipos de atividades podem estar presentes em definições de processos de workflow, mas em escalonamento a maioria das pesquisas concentra-se em ambientes onde ou todos os jobs são *preemptive* ou todos não o são. Embora a modelagem deste problema possa ser relativamente simples em termos de implementação para testes, não se conhece os resultados de tal amálgama de atividades *preemptive* e não-*preemptive* em um mesmo sistema.

4.2.6 A Natureza Dinâmica dos Sistemas de Workflow

Workflows são sistemas dinâmicos: enquanto existem várias instâncias de processo em andamento, novos casos chegam, e o sistema deve estar ciente deste fato. Problemas de impacto ainda desconhecido, como o fato de uma atividade não-*preemptive* estar sendo executada em um recurso, ou seja, tomando o recurso para si apenas, quando um caso urgente chega exatamente para o mesmo recurso, podem acontecer. Além disso, realizar

uma ordenação de atividades inicial e não levar em conta a presença desses novos casos pode causar deterioração do objetivo primário dessa ordenação, tornando-a pouco útil.

Em escalonamento, grande parte das pesquisas são focadas em situações estáticas, onde todos os jobs são conhecidos previamente, e uma vez gerada a ordenação, essa não muda. Há também pesquisas em situações dinâmicas, tais como [3, 5, 9, 22]. Como este dinamismo é também um aspecto considerado nesta pesquisa, a abordagem de decompor um problema de escalonamento dinâmico em uma série de problemas estáticos, como proposta por Raman e Talbot em [22] e utilizada por Bierwirth et al. em [5], foi utilizada, como será detalhado adiante.

4.2.7 O Mapeamento $m : n$ Entre Recursos e Atividades

Na literatura de escalonamento, um passo de execução de um job geralmente é feito apenas por uma máquina, ou seja, há o que podemos chamar de "mapeamento" $1 : 1$ entre passos de execução e máquinas. Ou então existem bancos de máquinas paralelas onde os jobs devem ser executados uma vez em cada um desses bancos, em qualquer de suas máquinas (ver definição de "identical machines in paralell" na seção 3.2.1), mas uma máquina pertence somente a um desses bancos, o que leva a um mapeamento $1 : n$ entre passos e máquinas. Em workflow as atividades e os recursos (que são os conceitos relativos a passos e máquinas em workflow) seguem um padrão diferente: uma atividade pode ser executada por mais de um recurso, mas um recurso também pode executar mais de um tipo de atividade, ou seja, esse mapeamento em workflow é $m : n$. As implicações desse fato são ainda desconhecidas.

Capítulo 5

Aplicando Escalonamento a Workflow

Como já visto, aplicar técnicas de escalonamento em sistemas de workflow não é tão simples. Por isso, algumas das características mencionadas no capítulo anterior foram escolhidas para serem estudadas. Este capítulo trata do desenvolvimento da pesquisa, detalhando como cada um de seus aspectos foi tratado.

5.1 O Problema

Em workflow é difícil saber exatamente quanto tempo cada atividade leva para ser executada, principalmente porque o fator humano, muito importante neste contexto, não é previsível. Além disso, quando o processo possui rotas alternativas dentro de si mesmo (*OR-splits*), a decisão de qual rota um caso específico deve seguir é tomada localmente, na atividade da qual as sub-rotas partem, ou seja, não se sabe qual o caminho exato que uma instância seguirá dentro de sua definição de processo. Em muitos casos as empresas nem mesmo guardam dados suficientes sobre seus próprios processos, inviabilizando modelagens mais precisas de sua situação. No melhor dos casos, existe uma (possivelmente) boa previsão para estes tempos de execução e rotas das instâncias. Junte-se a tudo isso o dinamismo das instâncias novas chegando ao sistema, enquanto outras estão no meio de sua execução e outras já em fase de término, e teremos um cenário típico de workflow.

Como já comentado, a maioria dos sistemas de workflow atuais utiliza a política FIFO para atribuir trabalho a seus participantes, sendo a política SIRO também ocasionalmente utilizada. Surge então o questionamento sobre a real efetividade desse tipo de política nesses sistemas. O problema torna-se realizar testes para verificar se há a possibilidade de melhora em alguns aspectos dos sistemas de workflow, como número de casos atrasados, com a aplicação de outras técnicas de ordenação.

Este trabalho apresenta um cenário de workflow que contém as características mencionadas anteriormente e estuda, através de simulação, o comportamento de diferentes técnicas de escalonamento dos casos que por ele passam. Para isso é feito um mapeamento desse cenário em um problema de escalonamento.

5.2 Metodologia

Para o desenvolvimento desse estudo, a metodologia adotada foi:

1. a definição de um cenário de workflow que contivesse as características desejadas para estudo;
2. o mapeamento do cenário em um problema de escalonamento, mais especificamente um *job shop*;
3. a definição e o estudo das técnicas de escalonamento a serem utilizadas neste cenário;
4. a definição de métricas para a avaliação destas técnicas;
5. a simulação de execução de processos no cenário proposto;
6. a análise dos resultados das simulações, para a efetiva avaliação do comportamento das técnicas de escalonamento segundo as métricas definidas, no cenário proposto;

O passo 3 envolveu tanto a pesquisa por técnicas já existentes quando a proposição de novas técnicas e a resolução dos problemas que estas pudessem vir a ter, como veremos adiante.

5.3 O Cenário Estudado

Para o estudo da relação workflow-escalonamento foi criado um cenário de workflow simples, representando uma definição de processos que contém as características que se deseja tratar. Este cenário está delineado na figura 5.1.

O cenário é composto de 4 atividades, identificadas por números de 1 a 4 respectivamente. Existe um *OR-split* partindo da atividade 1 para as atividades 2 e 3, ou seja, quando uma instância desse processo tem sua atividade 1 completada, ela é roteada ou para a atividade 2 ou para a atividade 3, com uma probabilidade de 50% de ir para uma ao invés da outra. Assim que esta instância sai da atividade 2 ou 3, é roteada para a atividade 4. Quando a instância termina a atividade 4, é considerada como um trabalho completo e sai do escopo do sistema.

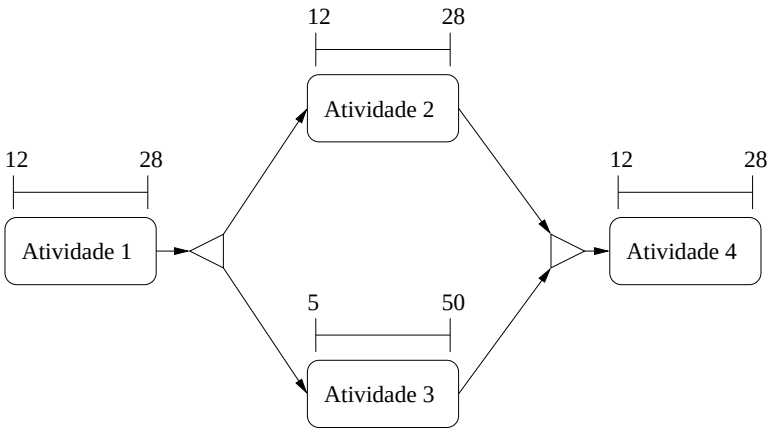


Figura 5.1: O cenário estudado: atividades, rotas e intervalos de tempo

O tempo de execução das instâncias em cada atividade é escolhido aleatoriamente através de uma distribuição uniforme sobre um intervalo. Para cada atividade, um intervalo de tempo é determinado. Os intervalos escolhidos para nosso cenário são $[12, 28]$ para as atividades 1, 2 e 4, e $[5, 50]$ para a atividade 3. Ou seja, se i é a atividade, j é a instância e $p_{i,j}$ é o tempo de processamento da instância j em i, então $p_{i,j} \in [12, 28]$ para $i \in \{1, 2, 4\}$ e $p_{i,j} \in [5, 50]$ para $i \in \{3\}$. A escolha destes valores para os limites dos intervalos segue a tendência de determiná-los livremente conforme a necessidade do problema, como o fazem [3, 4, 5, 12], por exemplo. Neste caso, desejava-se criar uma diferença entre o tempo que uma instância passa na atividade 2 e outra na atividade 3, "embaralhando" essas instâncias quando da sua chegada à atividade 4. Tais intervalos também estão demonstrados na figura 5.1.

5.3.1 O Mapeamento para um Job Shop

Para podermos aplicar as técnicas de escalonamento a esse cenário, deve-se mapeá-lo em um problema de escalonamento. Devido às suas características básicas, decidiu-se por um *job shop*. Como os conceitos dessa seção serão de fundamental importância para o restante do trabalho, a definição de um job shop juntamente com conceitos adicionais necessários será feita com mais detalhes.

Um *job shop* (*job shop scheduling problem*, ou *JSSP*) é definido como um conjunto J de n jobs que necessitam de processamento em uma ou mais máquinas pertencentes a um conjunto M. O número de máquinas em M é representado por m, e um elemento genérico de M é dado por i. Cada job j tem uma rota própria para seguir entre o conjunto de máquinas, então para j existe uma "ordem tecnológica" μ_j que representa a seqüência

de máquinas que este job deve seguir. O número total de passos de execução de j é representado por m_j. Cada k-ésimo passo de execução de j, $1 \leq k \leq m_j$, é definido como uma *operação* $o_{j(k)}$.

A ordem tecnológica μ_j pode ser considerada uma função que mapeia as operações de um job nas máquinas onde elas efetivamente acontecem. Deste modo, $\mu_j(k) = i$ significa que a operação $o_{j(k)}$ será executada na máquina i. Por exemplo, $\mu_1(3) = 2$ significa que o job 1 executará sua operação $o_{1(3)}$ (seu terceiro passo de execução) na máquina 2.

Um escalonamento em um job shop pode ser visto como uma tabela dos *starting times* (instante de tempo no qual uma operação começa sua execução) $t_{j(k)}$ das operações $o_{j(k)}$, respeitando a ordem tecnológica dos jobs [4]. Cada job j tem um tempo de processamento $p_{i,j}$ em uma máquina i de M. Em contextos dinâmicos, j também possui uma *release date* r_j que representa o instante de tempo no qual esse job chega ao sistema. O *completion time* (instante de tempo no qual a operação termina sua execução) de uma operação $o_{j(k)}$ é dado pelo seu *starting time* mais o seu tempo de processamento na máquina em questão, ou seja, $t_{j(k)} + p_{i,j}$, onde $i = \mu_j(k)$.

Para o cenário descrito, cada atividade é considerada uma máquina de um job shop, ou seja, $M = \{1, 2, 3, 4\}$. Os jobs são as instâncias dos processos, e as operações são a efetiva execução de uma instância em uma atividade. Existem duas possíveis rotas de execução para os jobs: as seqüências de máquinas (1,2,4) e (1,3,4). Os tempos de processamento dos jobs em cada máquina são retirados dos intervalos de tempo já mencionados, através de uma distribuição uniforme. As *release dates* são atribuídas de forma que uma carga de trabalho desejada para o sistema seja atingida (isso será detalhado mais adiante). Cada job j também possui um prazo para o término de sua execução, chamado de *due date* e representado por d_j. Essa *due date* é atribuída proporcionalmente ao tempo de processamento total do job j, multiplicando-o por um valor constante chamado *allowance factor*, e somando sua *release date*, para levar em conta que o job tem um prazo relativo ao momento de sua chegada ao sistema. Matematicamente temos

$$d_j = r_j + A \sum_{i=1}^{m} p_{i,j}$$

onde A é o *allowance factor*. Daqui para frente, o foco de estudo passa a ser este cenário de *job shop*.

5.3.2 Geração da Carga de Trabalho

A carga de trabalho, ou *workload* no termo em inglês, é considerada um aspecto crucial do desempenho de um sistema [4]. Também chamada de *work in process*, é definida como o número de tarefas que estão esperando para serem executadas no sistema [4, 17, 21]. Sendo assim, ela depende das taxas de chegada e saída de jobs ao/do sistema.

Para controlar a carga de trabalho do cenário em questão, usa-se um modelo proposto por Mattfeld e Bierwirth [4, 5]. A idéia do modelo é controlar os tempos entre a chegada de dois jobs consecutivos ao sistema. Se estes tempos forem maiores, os jobs chegarão mais espaçadamente entre si e o sistema trabalhará com mais folga, ou seja, com uma carga de trabalho menor. Se forem menores, haverá mais jobs no total, o que aumentará a carga de trabalho.

No modelo, o tempo médio entre a chegada de dois jobs consecutivos (λ) é dado pelo tempo de processamento esperado para um job qualquer ($\overline{p}$), o número de máquinas no sistema (m) e uma taxa de utilização desejada para essas máquinas (U). Se o valor de $\overline{p}$ for dado, então $\lambda = \overline{p}/mU$. Segundo Laguna [17], é possível observar empiricamente que muitos processos de chegada de jobs em situações reais podem ser caracterizados por uma distribuição exponencial. Então os tempos entre a chegada de dois jobs são gerados de acordo com uma distribuição exponencial cuja média é λ.

Para o cenário estudado, $\overline{p}$ é calculado primeiro calculando o tempo esperado de processamento em cada máquina, chamado aqui de $\overline{p_i}$, e que é baseado nos valores limitantes dos intervalos de tempo dos quais os tempos de processamento $p_{i,j}$ dos jobs são retirados. Para as máquinas 1, 2 e 4, este intervalo é $[12, 28]$, e portanto $\overline{p_1} = \overline{p_2} = \overline{p_4} = (12 + 28)/2 = 20$. Para a máquina 3, seguindo o mesmo princípio, tempos $\overline{p_3} = (5 + 50)/2 = 27.5$. Como exitem duas rotas possíveis para os jobs, pelas seqüências de máquinas (1,2,4) e (1,3,4), o tempo de processamento esperado para cada uma delas é $\overline{p_1}+\overline{p_2}+\overline{p_4} = 60$ para a primeira e $\overline{p_1}+\overline{p_3}+\overline{p_4} = 67.5$ para a segunda. Já que há uma probabilidade de 0.5 de um job seguir a primeira rota ao invés da segunda, então $\overline{p} = 0.5\times60+0.5\times67.5 = (60+67.5)/2 = 63.75$. Com este valor de $\overline{p}$, pode-se calcular λ para cada valor desejado de U.

Neste estudo, os valores de U considerados são 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85 e 0.95, que representam desde sistemas mais simples e de baixa carga de trabalho até sistemas complexos de alta carga.

5.3.3 Funções-Objetivo no Cenário

Para analisar o desempenho das técnicas de escalonamento no cenário em questão, dois tipos de objetivos de otimização foram definidos: os relacionados com as *due dates* (prazos) dos jobs e os relacionados com o tempo de processamento destes. Para o primeiro tipo, duas funções-objetivo foram analisadas: a porcentagem média de jobs atrasados e a porcentagem média de atraso. Do segundo tipo, analisou-se o tempo de processamento médio dos jobs no sistema. Vale lembrar que todas elas são, de uma forma ou de outra, relacionadas com o *completion time* C_j dos jobs, que é o instante de tempo onde o job j completa sua execução e sai do sistema. A seguir definimos cada uma dessas métricas.

O tempo médio de processamento de um job no sistema também é conhecido na

terminologia de escalonamento como o *processing time* médio do sistema. É uma métrica relativamente simples. É dado por

$$M_{\mathrm{pt}} = \frac{1}{n}\sum_{j=1}^{n}(C_j - r_j) \ . \tag{5.1}$$

Como o próprio nome indica, mede o tempo médio que um job leva para ser executado no sistema. Minimizar esta métrica significa fazer com que os jobs sejam completados mais rápido.

Para definir a porcentagem de jobs atrasados, tomamos a função *unit penalty* definida na equação 3.2 e calculamos o número de jobs atrasados no sistema como

$$n_{\mathrm{t}} = \sum_{j=1}^{n} U_j$$

e a porcentagem de jobs atrasados fica óbvia como

$$\overline{U_{\%}} = \frac{n_{\mathrm{t}}}{n} \ .$$

A porcentagem média de atraso no sistema é uma medida de quanto um job atrasado está atrasado no sistema. É calculado por uma razão entre o atraso médio de um job atrasado e o tempo médio de processamento de um job qualquer no sistema.

O atraso médio por job atrasado utiliza o conceito de *tardiness* definido na equação 3.1. É calculado como

$$\overline{T_{\mathrm{t}}} = \frac{1}{n_{\mathrm{t}}}\sum_{j=1}^{n} T_j \ . \tag{5.2}$$

Deste modo, a porcentagem média de atraso é dada por

$$\overline{T_{\%}} = \frac{\overline{T_{\mathrm{t}}}}{M_{\mathrm{pt}}} \tag{5.3}$$

onde M_{pt} é definido na equação 5.1

O resultado da equação 5.3 pode ser interpretado como a porcentagem do tempo de processamento médio dos jobs que um job atrasado gasta a mais para executar. Por exemplo, se a média de tempo de processamento do sistema é de 100 unidades de tempo, e temos uma porcentagem média de atraso de 25%, isso quer dizer que, em média, um job atrasado gasta $100 + 0.25 \times (100) = 125$ unidades de tempo em sua execução.

Quando o objetivo é fazer com que os jobs sejam executados o mais rápido possível, sem preocupações extras, o *processing time* médio é uma boa métrica. Já quando a preocupação principal é o número de jobs atrasados, ou quando cada job atrasado implica

uma penalidade fixa para a empresa, como uma reclamação de um cliente que deve ser respondida em no máximo 30 dias, a porcentagem de jobs atrasados é uma métrica mais adequada. E quando a quantidade de atraso dos jobs é o problema maior, como uma reunião que geralmente dura duas horas mas está dez minutos atrasada, a porcentagem média de atraso oferece mais vantagens, já que o tempo médio de processamento dos jobs (M_{pt}) normaliza cada caso: a reunião pode não estar tão atrasada já que dez minutos representam pouco tempo perto de suas duas horas de duração média.

5.4 Técnicas de Escalonamento em Job Shops

Muitas técnicas foram propostas para solucionar o problema de *job shop*. Uma lista destas técnicas pode ser extraída do trabalho de Jain e Meeran em [14], como por exemplo métodos eficientes para instâncias específicas do problema, técnicas *branch and bound*, heurísticas como as *bottleneck-based*, redes neurais, *simulated annealing*, *tabu search* e algoritmos genéticos. Para nossa pesquisa, escolheu-se trabalhar com *dispatching rules* bem conhecidas, visto que estas são relativamente fáceis de serem utilizadas e requerem pouco poder computacional, e além disso porque a regra FIFO é a mais utilizada em sistemas de workflow atuais [28]. Utilizou-se também algoritmos genéticos como uma possível solução para o problema.

5.4.1 Dispatching Rules

As *dispatching rules*, ou *priority rules*, são regras capazes de guiar a escolha dos jobs que esperam processamento no sistema. São baseadas em critérios que dizem respeito aos próprios jobs, como suas *due dates*, e também em critérios relativos ao conjunto do sistema, como a carga total de trabalho atual. Seu funcionamento é simples: quando uma máquina fica livre, o job na sua fila que mais se encaixa no critério da regra é processado.

Para esta pesquisa, algumas regras bem conhecidas foram escolhidas, e estão discriminadas abaixo.

- **FIFO** (First In First Out). Seleciona os jobs na ordem de sua chegada às filas de espera. Quem chegou primeiro é executado primeiro. É uma regra simples, e é a mais usada pelos sistemas de workflow atuais para despachar trabalho aos seus participantes.
- **SIRO** (Service In Random Order). Seleciona os jobs em ordem aleatória. Outra regra simples de ser implementada. Equivale a uma outra situação também existente em sistemas de workflow, onde o próprio recurso escolhe, dentre os casos disponíveis, qual ele irá executar.

- **EDD** (Earliest Due Date). Seleciona os jobs pela ordem de suas *due dates*. Quem tem a menor *due date* é executado primeiro. Uma regra também simples mas conhecida pela eficácia em minimizar jobs atrasados e atraso médio em sistemas de escalonamento.

- **SPT** (Shortest Processing Time). Seleciona os jobs na ordem de seu tempo de processamento no recurso em questão. O job que possui o menor tempo de processamento é executado primeiro. Também conhecida pela sua eficácia em minimizar o tempo de processamento total em um sistema de escalonamento. Para sistemas $1||\sum C_j$, ou seja, com uma máquina, onde se deseja minimizar a somatória dos *completion times* dos jobs, a política SPT é ótima. Esta situação se equivale ao escalonamento de processos em um microprocessador quando se deseja que o sistema responda rapidamente ao usuário, já que este tempo de resposta é uma importante medida da eficiência dos sistemas de computação. Portanto a política SPT também é conhecida na área de sistemas operacionais.

- **LSD** (Largest Successive Difference). Seleciona o job cujo tempo de processamento na máquina atual seja o menor, e o tempo de processamento na máquina seguinte seja o maior. Formalmente, o job j selecionado é aquele com o maior valor para a diferença $p_{k,j} - p_{i,j}$, onde $p_{i,j}$ é o seu tempo de processamento na máquina atual, e $p_{k,j}$ é seu tempo de processamento na máquina seguinte. Lawrence e Sewel [18] mostram que esta regra é eficiente na minimização do *makespan* (instante de tempo no qual o último job termina sua execução) em sistemas de escalonamento.

5.5 A Técnica Guess and Solve

As incertezas quanto aos tempos de execução e rotas dos jobs no sistema certamente levariam o problema para o domínio estocástico. Por isso uma modelagem para estas incertezas se faz necessária. Esta seção apresenta a técnica *guess and solve* proposta para este fim.

Lawrence e Sewell [18] argumentam que um possível meio de simplificar um problema estocástico é utilizar os valores esperados das variáveis nele presentes para representar o tempo de processamento dos jobs, e assim tratar o problema deterministicamente. A técnica *guess and solve* tem basicamente a mesma idéia, mas não opera com médias de distribuições. Consiste em realizar uma previsão controlada dos tempos de processamento e rotas dos jobs no sistema (a fase de *guess*), e depois resolver o problema determinístico resultante com uma técnica apropriada (a fase de *solve*). Assume-se que o sistema possui um *guesser* capaz de gerar previsões $p'_{i,j}$ para cada tempo de processamento $p_{i,j}$ dos jobs

do sistema com um erro máximo de f_{p}. Isso significa que a propriedade

$$f_{\mathrm{p}} \geq \left| \frac{p'_{i,j} - p_{i,j}}{p_{i,j}} \right|$$

vale para toda a previsão feita pelo *guesser*, onde $0 \leq f_{\mathrm{p}} \leq 1$. Este também é capaz de prever a rota que um job terá dentro do sistema com uma probabilidade de erro igual ao valor de f_{p}. Neste trabalho, três situações de erro para o *guesser* são consideradas: 10%, 20% e 30% de erro, ou seja, $f_{\mathrm{p}} \in \{0.1, 0.2, 0.3\}$.

O *guesser* representa uma abstração para várias técnicas que podem ser utilizadas para gerar previsões acerca das características dos jobs, tais como a utilização de *machine learning* [19], estatísticas baseadas em casos passados, ou mesmo a experiência de trabalho de alguns funcionários que lhes permite realizar tais predições. Mas o *guess and solve* exige que, qualquer que seja a técnica utilizada, ela deve possuir uma precisão (ou erro máximo) conhecida. Depois de realizadas tais previsões, o problema pode ser resolvido deterministicamente com uma técnica de escalonamento adequada. Nesta pesquisa, a escolhida foi a utilização de algoritmos genéticos, e para a otimização do *processing time* médio, também foram utilizadas algumas *dispatching rules*.

5.6 Algoritmos Genéticos

Segundo Mitchell [19], um algoritmo genético, ou simplesmente AG, é um método de aprendizado em inteligência artificial motivado por uma analogia com a evolução biológica. Seu ponto de partida é um conjunto inicial de hipóteses, chamada população, e através de sucessivas recombinações e mutações destas hipóteses gera uma nova população, que tenta "explicar" melhor o problema estudado. As hipóteses são, na verdade, representações de possíveis soluções para tal problema, podendo ser cadeias de bits, vetores de números inteiros, ou qualquer outra maneira de respresentação que se se faça pertinente. Essa representação é um aspecto importante para um AG, pois quando adequadamente realizada, contribui muito para resultados melhores por parte do algoritmo.

Um algoritmo genético realiza uma busca por um espaço de possíveis hipóteses para identificar a melhor dentre elas. Para um AG, a "melhor hipótese" é aquela que otimiza uma métrica numérica predefinida para o problema em questão, e que é chamada *fitness* da hipótese. Quanto maior o seu *fitness*, melhor uma hipótese é considerada.

Em termos gerais, um AG gera uma população inicial de hipóteses P com $|P|$ elementos, avalia $fitness(p)$ para todo elemento $p \in P$, e depois entra em uma iteração onde, enquanto uma condição de parada não é atingida, realiza-se: (i) uma substituição de uma parte de P por elementos gerados por um processo de recombinação (o restante dos elementos mantêm-se inalterados); (ii) um processo de mutação dos elementos de P;

(iii) uma nova avaliação de $fitness(p)$ para todo $p \in P$. A recombinação é feita por um operador de crossover, que geralmente seleciona dois elementos de P, chamados *pais*, e gera um ou mais elementos derivados destes, chamados de *filhos*. Essa seleção ocorre probabilisticamente, sendo que quanto maior o *fitness* de um elemento, maior a probabilidade de ele ser escolhido como pai. Comparando cada hipótese com uma seqüência genética de um ser vivo, encontramos a analogia dos AGs com a evolução biológica: teoricamente, quanto melhores geneticamente forem os pais, melhores as chances de que gerem filhos mais evoluídos. Assim, quanto maior for o *fitness* dos pais, melhores são suas chances de gerarem hipóteses-filhas com *fitness* superior. Por último, escolhem-se alguns indivíduos da nova população para sofrerem "mutação", um processo que realiza pequenas modificações nas hipóteses para provocar pequenas variações de seus valores de *fitness*. Comparando o crossover com a mutação, pode-se dizer que o primeiro realiza uma busca de "longa distância" no espaço de soluções, enquanto que a segunda realiza uma busca curta.

Cada iteração do algoritmo é chamada de uma *geração*, já que uma nova população derivada da antiga é criada. A condição de parada pode ser um número máximo de gerações para o algoritmo, ou quando não é mais possível encontrar uma hipótese com maior *fitness*, ou ainda uma combinação dessas duas. O retorno do AG é a melhor hipótese por ele encontrada, ou seja, a sua melhor resposta para o problema. No caso deste trabalho, a melhor hipótese retornada pelo AG sofre um processo de decodificação, cujo resultado é uma tabela contendo todos os *starting times* $t_{j(k)}$ de cada operação dos jobs. Essa decodificação será explicitada adiante.

5.7 Implementação do AG para uso com o *Guess and Solve*

Para utilizar AGs com o *guess and solve* implementou-se um misto entre o AG apresentado por Mitchell em [19] e o processo de mutação de Moscato et al. em [12], juntamente com adequações necessárias ao problema de escalonamento de um *job shop*. Cada hipótese h_k presente na população do AG codifica uma possibilidade de escalonamento para o sistema, sendo o termo "representação" um outro nome para cada hipóteste.

Três implementações de algoritmos genéticos foram utilizadas: uma para o AG que otimiza o número de jobs atrasados, outra para o que otimiza a porcentagem média de atraso, e outra para o AG que otimiza o *processing time* médio dos jobs. Isso porque as evidências encontradas em [3, 4, 5, 14] apontam para o problema de minimizar o número de jobs atrasados como mais factível de resolução por AGs, enquanto que na minimização do *processing time* é sensivelmente mais complicado obter-se resultados satisfatórios com

estes algoritmos. Além disso, diferenças na função objetivo também podem afetar a definição da função de *fitness* do AG, que é o caso entre o primeiro e o segundo AGs. As diferenças entre as implementações são também tratadas no texto que segue. Para facilitar as referências, o AG para o primeiro problema será chamado de AG1; para o segundo, AG2; e para o terceiro, AG3.

5.7.1 A Representação de um JSSP para o AG

A representação utilizada para as hipóteses é a apresentada por Bierwirth et al. em [3, 4, 5]. Nela, cada hipótese é dada por uma permutação de todas as operações $o_{j(k)}$ de todos os jobs presentes no sistema. Suponhamos um job shop com dois jobs e duas máquinas. Ambos os jobs executam em todas as máquinas. Portanto, temos as operações $o_{1(1)}, o_{1(2)}, o_{2(1)}$ e $o_{2(2)}$. Qualquer permutação destas pode ser considerada uma representação de um schedule para o sistema.

Uma permutação destas não possui todas as informações necessárias para escalonarmos as operações diretamente delas, sendo necessário um algoritmo de decodificação, que recebe como entrada uma hipótese e gera como saída um schedule. Utilizou-se então dois algoritmos de decodificação dessas permutações, também mostrados por Bierwirth et al. em [3, 4, 5], sendo um para o AG1 e AG2, e outro para o AG3. O primeiro é descrito na tabela 5.1, e sua saída são schedules *non-delay*.

O segundo algoritmo, utlizado para o AG3, envolve uma adaptação para aumentar o espaço de busca do AG para schedules que estão em um domínio pertencente tanto aos *non-delay* como *active*. Esta adaptação é feita como sugerem Bierwirth e Mattfeld em [5], e consiste na introdução de um parâmetro $\delta \in [0, 1]$. Nos extremos, quando $\delta = 0$, o algoritmo de decodificação gera schedules *non-delay*; quando $\delta = 1$, apenas schedules *active* são gerados; entre esses dois valores, os schedules gerados são híbridos de *non-delay* e *active*. Esta operação é também chamada de "decodificação híbrida". Tal algoritmo é dado na tabela 5.2.

Os dois algoritmos garantem a geração de *feasible schedules*, ou seja, escalonamentos que não violam as regras estabelecidas para o cenário. Isso elimina um passo adicional de análise e eliminação de escalonamentos que não são *feasible*.

5.7.2 A Geração da População Inicial

A população inicial do AG é gerada por um processo simples que parte de uma primeira permutação ρ_0 de operações colhida junto ao sistema. Uma segunda permutação é gerada escolhendo-se aleatoriamente duas operações de ρ_0 e trocando suas posições. Neste momento a população inicial possui dois elementos válidos. Agora escolhe-se aleatoriamente um deles para servir de origem a uma nova representação pelo processo já mencionado. No

1. Construir o conjunto A de todas as primeiras operações dos jobs, $A = \{o_{j(1)} | 1 \leq j \leq n\}$.

2. Determinar o menor *starting time* t' de todas as operações de A, $t' \leq t_{j(k)}$ para todo $o_{j(k)} \in A$.

3. Construir o conjunto B de todas as operações cujo *starting time* é igual a t', $B = \{o_{j(k)} \in A | t_{j(k)} = t'\}$.

4. Selecionar a operação $o^*_{j(k)}$ de B que ocorre mais à esquerda da permutação que está sendo decodificada.

5. Remover a operação $o^*_{j(k)}$ de A, $A = A - \{o^*_{j(k)}\}$.

6. Se $o^*_{j(k+1)}$ existir, inserí-la em A, $A = A \cup \{o^*_{j(k+1)}\}$.

7. Se $A \neq \emptyset$ volte ao passo 2, senão pare.

Tabela 5.1: Algoritmo de decodificação para o AG1 e o AG2

próximo passo haverá três indivíduos válidos, e assim sucessivamente até que se complete o número de indivíduos total da população inicial. O algoritmo de geração da população inicial é apresentado na tabela 5.3.

5.7.3 Operador de Crossover

O operador de crossover utilizado é bem conhecido, chamado *Order-Based Crossover*, ou OX. Como muitos outros operadores, seleciona duas representações pais e gera uma nova representação filha a partir daquelas. Suponhamos as duas permutações pais ρ_1 e ρ_2, e ρ' a permutação filha. O algoritmo desse operador é dado na tabela 5.4.

A figura 5.2 ilustra os passos 2 e 3 do algoritmo de crossover. Neste exemplo, dois indivíduos da população, que são permutações das operações A, B, C, D, E, F, G, H e I foram escolhidos como os pais. Considerando as posições de ρ_1 e ρ_2 começando em 1, $i_1 = 4$ e $i_2 = 7$.

A porcentagem da população substituída pelo crossover é 60%, um valor bem conhecido e utilizado. Testes com um outro operador de crossover, chamado por Bierwirth e Mattfeld, em [5], de PPX, também foram realizados, mas este não obteve resultados melhores, sendo portanto abandonado como alternativa de implementação.

1. Construir o conjunto A de todas as primeiras operações dos jobs, $A = \{o_{j(1)} | 1 \leq j \leq n\}$.

2. Determinar uma operação o' de A com o menor *completion time* possível $t' + p'$, $o' = o_{j(k)} | t' + p' \leq t_{j(k)} + p_{\mu_j(k),j}$ para todo $o_{j(k)} \in A$.

3. Determinar a máquina i onde o' executa e construir o conjunto B de todas as operações em A que são processadas em i, $B = \{o_{j(k)} | \mu_j(k) = i\}$.

4. Determinar uma operação o'' de B com o menor starting time possível t'', $o_{j(k)} = o'' | t'' = t_{j(k)}$ para todo $o_{j(k)} \in B$.

5. Remover operações de B de acordo com o parâmetro δ tal que $B = \{o_{j(k)} | t_{j(k)} < t'' + \delta((t' + p') - t'')\}$.

6. Selecionar a operação $o^*_{j(k)}$ de B que ocorre mais à esquerda da permutação que está sendo decodificada e removê-la de A, $A = A - \{o^*_{j(k)}\}$.

7. Se $o^*_{j(k+1)}$ existir, inserí-la em A, $A = A \cup \{o^*_{j(k+1)}\}$.

8. Se $A \neq \emptyset$ volte ao passo 2, senão pare.

Tabela 5.2: Algoritmo de decodificação para o AG3

5.7.4 Mutação

Dois esquemas de mutação foram utilizados na pesquisa, um para o AG1 e AG2, e outro para o AG3. Para o AG1 e AG2, a mutação foi implementada como sugerida por Moscato et al. em [12]: todas as hipóteses originárias do operador de crossover sofrerão mutação com uma probabilidade mut_{p}. Assim que uma hipótese filha ρ' é gerada pelo crossover, um número aleatório c entre 0 e 1 é escolhido e, caso $c >= mut_{\mathrm{p}}$, a hipótese sofre mutação. Já para o AG3, o processo é aquele descrito por Mitchell [19] e utilizado por Bierwirth e Mattfeld [5]: uma porcentagem p da população do AG3 sofrerá mutação, ou seja, escolhem-se aleatoriamente $\lfloor p|P| \rfloor$ elementos da população para sofrerem mutação. Em ambos os casos, o processo de mutação consiste em realizar a troca de posições de duas operações de uma hipótese. Novamente escolhem-se estas operações aleatoriamente.

Aqui também se utiliza um valor bem conhecido para a porcentagem/probabilidade de mutação, de 0.1 (10%).

1. Construir o conjunto P dos elementos da população inicial, e que contém como primeiro elemento a permutação ρ_0 extraída do sistema, $P = \{\rho_0\}$.
2. Selecionar aleatoriamente um elemento ρ' de P.
3. Selecionar aleatoriamente duas operações o' e o'' de ρ' e trocar suas posições.
4. Se indivíduos suficientes para preencher a população inicial já foram gerados, pare, senão volte ao passo 2.

Tabela 5.3: Algoritmo de geração da população inicial para os AGs

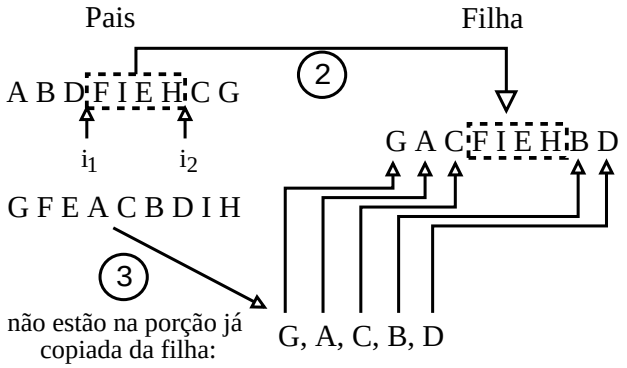


Figura 5.2: Exemplo do operador de crossover OX

5.7.5 Fitness

Neste caso também três funções para a avaliação do *fitness* das hipóteses foram utilizadas, uma para o AG1, outra para o AG2, e outra para o AG3. Para o AG1, que minimiza o número de jobs atrasados, a função de *fitness* foi definida como

$$\text{fitness}_1(h_k) = \frac{1}{1 + \text{nlate}(h_k)}$$

sendo $\text{nlate}(h_k)$ o número de jobs atrasados segundo a codificação de h_k. Para o AG2, que minimiza a porcentagem média de atraso, a função de *fitness* escolhida foi

$$\text{fitness}_2(h_k) = \frac{1}{1 + \overline{\overline{T_\text{t}}}}$$

1. Selecionar aleatoriamente duas posições i_1 e i_2 de ρ_1.

2. Copiar as operações da permutação ρ_1 que estão entre as posições i_1 e i_2 para as mesmas posições de ρ', ou seja, $\rho'[k] = \rho_1[k]$ para $i_1 \leq k \leq i_2$.

3. Percorrer ρ_2 da esquerda para a direita, copiando os elementos que não aparecem na porção já copiada de ρ_1, para as posições que ainda não foram preenchidas em ρ' (que são aquelas entre 1 e $i_1 - 1$ e entre $i_2 + 1$ e $|\rho_2|$), na ordem em que elas aparecem em ρ_2, ou seja, para $1 \leq k \leq |\rho_2|$, se $\rho_2[k] \neq \rho'[s]$ para todo $i_1 \leq s \leq i_2$, então $\rho'[l] = \rho_2[k]$, sendo que o índice l primeiro percorre o intervalo $[1, i_1 - 1]$ e em seguida $[i_2 + 1, |\rho_2|]$.

4. Inserir ρ' na nova população que está sendo gerada.

Tabela 5.4: Algoritmo do operador de crossover OX

onde $\overline{T_t}$ é o atraso médio por job atrasado, definido na equação 5.2. Para o AG3, que minimiza o *processing time* médio dos jobs, a função de *fitness* é

$$\text{fitness}_3(h_k) = \frac{1}{\frac{1}{n}\sum_{j=1}^{n}(C_j - r_j)}$$

sendo n o número de jobs que a hipótese h_k codifica. Deste modo, $\text{fitness}_3(h_k)$ é 1 dividido pelo tempo de processamento médio dos jobs da hipótes h_k.

5.8 Escalonamento Dinâmico com Algoritmos Genéticos

Em contextos dinâmicos, o que é o caso deste trabalho, os jobs a serem processados não são conhecidos de antemão; eles chegam ao sistema em instantes específicos de tempo, já definidos aqui como suas *release dates* r_j, e apenas depois dessas release dates serem atingidas é que o sistema toma conhecimento da existência do job. Simplesmente incluir o novo job em um escalonamento já gerado para o sistema pode deteriorar demais este último. Além disso, há uma limitação quanto aos AGs: devido ao seu modo de operação, eles somente são capazes de resolver um problema de escalonamento estático, o que quer dizer que eles precisam conhecer todos os jobs a serem escalonados.

Uma possível solução é decompor o problema de escalonamento dinâmico em uma série de problemas estáticos com os quais um AG possa trabalhar. Essa decomposição é

proposta por Raman e Talbot em [22], e usada por Bierwirth et al. em [3, 5]. A idéia é que, a cada chegada de um novo job ao sistema, um novo problema de escalonamento estático seja criado, resolvido e implementado. Isso permite o escalonamento estático ao sistema de tempos em tempos [22].

Em termos formais, tal método considera inicialmente um problema de escalonamento estático P_0, que é composto de todas as operações dos jobs que devem ser escalonados pela primeira vez no sistema. Esta primeira vez aqui é tratada como o instante de tempo t_0. O problema P_0 é resolvido normalmente e a sua solução é dada por uma tabela dos *starting times* $t_{j(k)}$ das operações $o_{j(k)}$.

Os *starting times* das operações são dados pelo instante de tempo mais cedo possível no qual uma operação $o_{j(k)}$ pode começar sua execução. Esse valor é o máximo entre os *completion times* (i) da operação que precede $o_{j(k)}$ na ordem tecnológica do job j, denotada por $o_{j(k-1)}$, e (ii) da operação $o_{h(l)}$ que precede a execução de $o_{j(k)}$ na mesma máquina que esta executa. Esta situação pode ser expressa matematicamente pela equação

$$t_{j(k)} = \max\{t_{j(k-1)} + p_{\mu_j(k-1),j}, t_{h(l)} + p_{\mu_h(l),l}\} \ . \tag{5.4}$$

Se a operação $o_{h(l)}$ não existe, então o segundo termo da equação passa a ser 0. Se $o_{j(k)}$ for na verdade a primeira operação de um job ($o_{j(1)}$), o que implica que $o_{j(k-1)}$ não existe, então considera-se a *release date* do job j, e o *starting time* de $o_{j(1)}$ é dado por

$$t_{j(1)} = \max\{r_j, t_{h(l)} + p_{\mu_h(l),l}\} \ . \tag{5.5}$$

Escalona-se as operações presentes em P_0 de acordo com estas equações e então já se pode implementar este primeiro resultado no sistema.

Quando um novo job chega ao sistema em um instante de tempo t_1, um novo problema estático P_1 é criado a partir de P_0. Primeiramente remove-se de P_0 todas as operações que já completaram sua execução até o instante t_1, ou seja, aquelas para as quais vale a propriedade $t_{j(k)} + p_{\mu_j(k),j} \leq t_1$, já que estas já deixaram o sistema e não tomarão mais parte em nenhum esforço de escalonamento. Em segundo lugar, é necessário recalcular os *release dates* dos jobs que tiveram operações removidas de P_0, mas ainda possuem outras para serem escalonadas. Esse cálculo é feito com base na equação

$$r_j = \max_{1 \leq k \leq m_j} \left\{ t_{j(k)} + p_{\mu_j(k),j} | t_{j(k)} < t_1 \right\}$$

onde m_j, como já visto, é o número de operações restantes do job j no sistema.

O terceiro passo está em identificar aquelas operações que iniciaram sua execução antes de t_1 mas ainda não a completaram até a chegada do novo job. Isso significa identificar as operacões para as quais vale a propriedade $t_{j(k)} < t_1 < t_{j(k)} + p_{\mu_j(k),j}$. Nestes casos, a máquina i que está executando uma dessas operações não estará disponível em t_1, então

um tempo de setup para essa máquina passa a ser considerado. Esse tempo de setup é calculado como

$$s_i = \max\left\{\max_{1\leq j\leq n}\left\{t_{j(k)} + p_{\mu_j(k),j} | t_{j(k)} < t_1\right\}, t_1\right\} \ .$$

Finalmente, todas as operações dos novos jobs, que chegaram ao sistema no instante, t_1 são adicionadas ao problema P_1. Neste ponto, P_1 constitui um novo problema de escalonamento estático e pode ser resolvido e implementado. Agora os tempos de setup das máquinas são levados em consideração, e caso uma operação $o_{j(k)}$ seja a primeira a ser executada por uma máquina i, então seu *starting time* é dado por

$$t_{j(k)} = \max\{t_{j(k-1)} + p_{\mu_j(k-1),j}, s_i\} \ .$$

Se uma operação é a primeira a ser executada em uma máquina i e também é a primeira operação de um job j (não necessariamente a primeira operação dentre todas de j, mas sim a sua primeira operação no problema corrente), então seu *starting time* é calculado por

$$t_{j(k)} = \max\{r_j, s_i\} \ .$$

As outras operações têm seus *starting times* calculados pelas equações 5.4 e 5.5.

Até que mais jobs novos cheguem em um tempo t_2, todas as operações de P_1, já escalonadas, podem ser implementadas no sistema. Repetindo este processo depois de cada chegada de novos jobs, o sistema é reescalonado periodicamente [22].

5.8.1 A Decomposição no Cenário Estudado

Na prática, a decomposição é uma maneira de coletarmos o estado do sistema no momento em que um novo job chega a ele, como se lhe tirássemos uma "fotografia" interna, e utilizar este estado para uma operação de escalonamento estático. No cenário aqui estudado, quando um novo job chega, este é colocado em espera na fila da máquina 1. Considera-se que os jobs que estão na fila desta máquina possuem 3 operações no sistema: a primeira na máquina 1, a segunda na máquina 2 ou 3 (dependendo da previsão de rota feita pelo *guesser*), e a terceira na máquina 4; um job na fila das máquinas 2 ou 3 terá duas operações, uma na máquina 2 ou 3 respectivamente, e outra na máquina 4; por último, um job na fila da máquina 4 terá apenas uma operação, nesta mesma máquina. Isso vale apenas para os jobs que estão em espera nas filas. Quando existem jobs que estão executando, o que se faz é considerar apenas as suas operações subseqüentes, pois a atual já está implementada. Mas agora estas terão *release dates* que equivalem ao tempo necessário para que as operações atualmente em processamento sejam completadas.

Tomemos a figura 5.3 como exemplo de um possível estado do sistema em um instante de tempo t qualquer, no qual o job 9 acaba de chegar. Os jobs 5 e 9 esperam na fila

da máquina 1; o job 3 espera pela máquina 2; os jobs 4 e 8 esperam pela máquina 3, e por último o job 1 espera pela máquina 4. Também estão sendo executados os jobs 6, 7 e 2 respectivmente nas máquinas 1, 3 e 4. Os jobs 6 e 7 terminarão suas execuções nos tempos t_6 e t_7. Não há nenhum job executando na máquina 2. Para este estado, a tabela 5.5 mostra, para cada job, as suas operações e respectivas *release dates*.

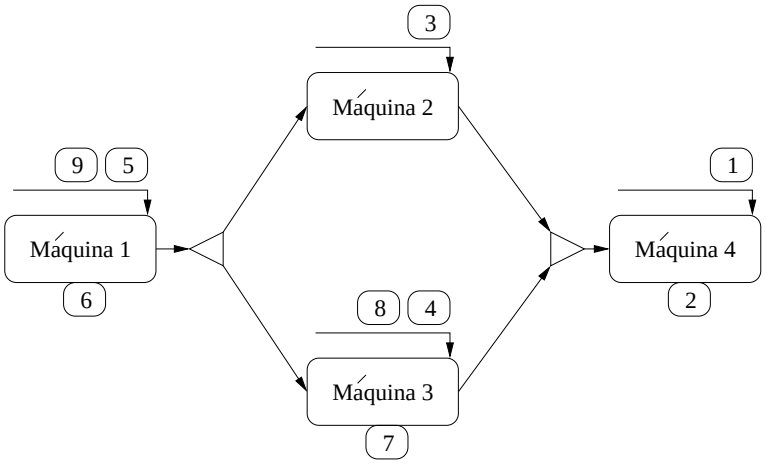


Figura 5.3: Exemplo de um possível estado do sistema

Job	**Operações**	r_j
1	$o_{1(1)} = 4$	0
2	–	–
3	$o_{3(1)} = 2$, $o_{3(2)} = 4$	0
4	$o_{4(1)} = 3$, $o_{4(2)} = 4$	0
5	$o_{5(1)} = 1$, $o_{5(2)} = 2$, $o_{5(3)} = 4$	0
6	$o_{6(1)} = 3$, $o_{6(2)} = 4$	$t_6 - t$
7	$o_{7(1)} = 4$	$t_7 - t$
8	$o_{8(1)} = 3$, $o_{8(2)} = 4$	0
9	$o_{9(1)} = 1$, $o_{9(2)} = 3$, $o_{9(3)} = 4$	0

Tabela 5.5: Operações e suas *release dates*, relativas aos jobs presentes no sistema

Um conjunto contendo as operações da tabela 5.5 pode ser considerado uma representação válida do estado do sistema no tempo t. Como veremos mais adiante, uma permutação destas operações será utilizada para representar uma possibilidade de escalonamento para o sistema, e conseqüentemente, para representar uma hipótese para o

algoritmo genético.

5.8.2 Incertezas Quanto aos Tempos de Processamento e Rotas: Ajustando o Dinamismo do Sistema

Como as previsões sobre os tempos de execução e rotas feitas pelo *guesser* são os únicos dados que se têm previamente sobre os jobs, todo o escalonamento é feito baseado nestas previsões. Mas quando o resultado da operação de escalonamento é implementado, os tempos de processamento e rotas reais são trazidos à tona, e algumas inconsistências podem ocorrer durante a passagem dos jobs pelo sistema. Por exemplo, um job j pode ser escalonado para começar sua execução em uma máquina i em um instante de tempo t_0, terminá-la em um instante t_1 e imediatamente começar sua execução em outra máquina. Mas o *guesser* fez uma previsão $p'_{i,j} < p_{i,j}$. Espera-se que o job saia de i em $t_0 + p'_{i,j} = t_1$. Mas o *completion time* real de j em i ($C_{i,j}$) é $t_0 + p_{i,j} > t_1$, e então o job deve começar a executar na próxima máquina em t_1 mas ainda não terminou seu processamento em i.

Para resolver as inconsistências causadas pelas incertezas nos tempos de execução dos jobs, uma adaptação à técnica de decomposição temporal do *job shop* foi realizada. Ela consiste em um novo passo após a decodificação das hipóteses dos AGs por um dos algoritmos das tabelas 5.1 e 5.2. A decodificação gera uma tabela contendo todos os *starting times* de cada job em cada máquina. Esta tabela é então usada para criar o que aqui se chama de "listas de ordem" ol_i para cada máquina i, que representam a ordem exata na qual os jobs serão executados nestas.

Quando um job chega para execução em uma máquina i, ele é posto em sua fila. Sempre que a máquina se torna livre, ela procura em sua fila o job cujo número é o que ocupa a primeira posição em ol_i. Se essa busca obtém sucesso, tal job é executado e seu número é removido de ol_i; senão, a máquina espera o job correto chegar. Se um job chega e a máquina já está livre, ela verifica se este é o job pelo qual ela está esperando. Se for, este é processado; senão o job é posto na fila da máquina. Esta alteração, embora simples, é conveniente pois permite que o sistema tenha um fluxo natural de jobs sem ter de detectar inconsistências e recalcular os *starting times* dos jobs quando alguma é descoberta.

A geração das listas de ordem é simples partindo da tabela de *starting times*. Percorre-se todas as posições da tabela e utiliza-se um algoritmo de ordenação para colocar os números dos jobs correspondentes nas listas de ordem corretas, em ordem crescente de seus *starting times*.

Outro tipo de inconsistência pode ocorrer devido ao esquema de listas de ordem, quando o *guesser* prevê que o job j seguirá uma rota, por exemplo a máquina i, mas na verdade ele segue outra, digamos, a máquina k. Como j não está presente em ol_k, esperará

eternamente na fila desta máquina e não será executado. E, pior, como ele consta em ol_i mas nunca irá para essa máquina, quando seu número for primeiro da lista de ordem a máquina i ficará eternamente esperando por ele e deixará de executar os outros jobs que chegam a ela. Isso é resolvido considerando que, quando esse tipo de inconsistência ocorrer, o job j será colocado em último lugar na fila e na lista de ordem da máquina k, e sua previsão incorreta de rota será corrigida, para que ele possa aproveitar-se de um escalonamento subseqüente causado pela chegada de um novo job.

5.9 Simulação

A simulação por computador é um eficiente meio de se estudar processos e prever seu desempenho [17]. Com ela, pode-se analisar vários aspectos de um processo de maneira quantitativa, fornecendo um panorama dinâmico de seu funcionamento, tanto interno (o comportamento dos jobs enquanto são executados) quanto externo (o *throughput* total do sistema, o tempo de espera dos clientes pela concretização do serviço prestado, entre outros). Torna-se então uma valiosa ferramenta para o estudo de processos (ou aspectos específicos desses processos) alternativos, em busca da melhor configuração para resolver o problema que se tem em mãos. Laguna [17] afirma que a técnica da simulação é muito útil no contexto de modelagem de processos de negócio pois este é um problema de tomada de decisões onde:

- o desenvolvimento de um modelo matemático para um determinado processo pode ser muito difícil ou até mesmo impossível;
- o desempenho do processo depende de uma ou mais variáveis aleatórias
- sua dinâmica é demasiadamente complexa;
- o comportamento do processo deve ser observado durante um período de tempo para que se possa validar seu design;
- uma demonstração gráfica dos resultados simulados torna-se muito útil;

A simulação também é capaz de cobrir certas ineficiências que geralmente passam despercebidas até que o sistema venha a entrar em operação [17]. Um gargalo pouco óbvio ou uma necessidade de manter estoques acima do esperado podem ser previstos, deste modo minimizando custos de reengenharia no futuro.

Existem diferentes tipos de modelos para simulação. O foco deste trabalho, no entanto, é na chamada *discrete-event simulation*, ou *simulação com eventos discretos*. Nela trata-se da evolução dinâmica de um sistema, através de uma representação na qual as variáveis

mudam de valor imediatamente, em instantes separados e específicos de tempo. Esses pontos são chamados de *eventos*. Ou seja, um evento é uma ocorrência instantânea que pode alterar o estado do sistema.

Para os testes necessários, implementou-se um simulador inicial para as primeiras análises, que evoluiu para outros simuladores quando da necessidade de adaptação para novos casos de teste. Escritos na linguagem Java, os programas gerados foram executados em plataformas Red Hat Linux 7.3, 8.0, 8.3 e 9.0. Internamente, todas as variáveis que guardam valores de tempo, tais como os tempos de processamento dos jobs, ou o tempo global da simulação, são números inteiros. Isso torna os simuladores independentes de tempo real, e portanto questões como a velocidade dos computadores onde os testes foram executados não influenciam o resultado final obtido.

Capítulo 6

Parametrização dos Testes e Resultados Obtidos

Este capítulo descreve a parametrização das diversas condições de simulação, bem como discute os resultados obtidos em cada análise.

6.1 Simulação e Testes

De acordo com as métricas de otimização definidas para este estudo, existem três cenários a serem testados: um onde se tenta otimizar a porcentagem de jobs atrasados, outro onde se quer otimizar o atraso médio dos jobs, e um terceiro onde o objetivo é otimizar o *processing time* médio dos jobs. Para cada cenário, um subconjunto das técnicas de escalonamento mencionadas na seção 5.4 foi escolhido para ser avaliado.

Para o primeiro e o segundo cenário, escolheu-se as *dispatching rules* EDD-FIRST e EDD-ALL, duas variantes da regra EDD. Na EDD-FIRST, os jobs são selecionados em ordem crescente de suas *due dates* apenas na primeira máquina, enquanto que nas outras a política de escalonamento é a FIFO. Já na EDD-ALL, todas as máquinas possuem suas filas regidas pela política EDD. Para o terceiro cenário, escolheu-se as regras SPT e LSD. As regras FIFO e SIRO estão presentes em todas as simulações pois formam a base de referência para os resultados das outras técnicas já que, como mencionado anteriormente, são as políticas mais comuns nos sistemas de workflow atuais. Os algoritmos genéticos também são aplicados a todos os casos de simulação, cada um com sua implementação específica voltada ao problema que devem resolver.

Como as regras SPT e LSD, e os AGs, utilizam de alguma forma dados sobre os tempos de execução dos jobs em cada máquina, e nas simulações somente possuímos esses dados provenientes do *guesser*, então cada uma delas é usada com as três versões deste último, ou seja, com 10%, 20% e 30% de erro. Assim, o *guess and solve* é realizado com

essas três técnicas de escalonamento, as quais também foram utilizadas em uma forma "pura", ou seja, quando conhecem os tempos de processamento exatos dos jobs, tanto para testes quanto para fins de comparação. Para referência, a regra SPT será chamada de SPT-PURE quando os tempos de execução dos jobs forem exatos, não provenientes do *guesser*; SPT-10, SPT-20 e SPT-30, quando esses valores vierem de *guessers* com 10%, 20% e 30% de erro, respectivamente. A mesma nomenclatura será aplicada à regra LSD e aos AGs, ou seja, teremos LSD-PURE, LSD-10, LSD-20, LSD-30, AG-PURE, AG-10, AG-20 e AG-30.

6.2 Resultados

Cada caso de simulação conta com um conjunto de 100 jobs que devem passar pelo sistema. Um mesmo conjunto é testado para todas as técnicas de escalonamento aplicadas a cada cenário, e os resultados de número de jobs atrasados, atraso médio e *processing time* obtidos por cada técnica são colhidos. Estes são analisados pelos valores médios obtidos em cada quesito, e também através de um teste estatístico chamado de *pairwise t-test*, para verificar se os resultados obtidos pelas outras técnicas podem ser considerados estatisticamente iguais aos resultados obtidos pela regra FIFO. Para todos os cenários, 50 conjuntos diferentes de jobs são testados. Apresenta-se a seguir esta análise em relação a cada função-objetivo estudada.

6.2.1 Porcentagem Média de Jobs Atrasados

Os resultados relativos à porcentagem média de jobs atrasados estão na tabela 6.1. Para apresentar os resultados do *pairwise t-test*, os dados desta tabela foram combinados com o resultado numérico do teste para formar a tabela 6.2. Nela existem três valores: um traço (-), *sim* e *não*. O traço quer dizer que a técnica em questão não é estatisticamente diferente da regra FIFO com 95% de confiança; *sim* indica que a técnica é melhor que FIFO, e *não* indica que é pior, também, com 95% de confiança.

Os AG-PURE e AG-10 são as melhores técnicas para taxas de utilização de 0.65 a 0.95. Os AG-20 e AG-30 também dão resultados melhores que a melhor *dispatching rule* para o mesmo intervalo de U, mas crescentemente pioram em relação ao AG-PURE e ao AG-10. Um erro de 10% não parece ser um grande problema para o AG-10, e portanto um *guesser* com esse erro mostra-se como uma ótima técnica de aproximação para uma solução baseada em algoritmos genéticos. Na verdade, a técnica *guess and solve* com AGs mostrou bons resultados mesmo com o maior erro testado, para esta função-objetivo.

Um resultado curioso vem da regra SIRO: ela é pior que EDD-ALL com U de 0.15 a 0.65, mas quando a utilização chega em 0.75, ela dá resultados muito melhores. Um estudo

Técnica	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
FIFO	0.02	0.28	0.44	2.40	8.52	22.14	44.30	59.58	74.40
EDD-FIRST	0.02	0.02	0.10	1.60	6.24	18.92	41.42	58.18	73.72
EDD-ALL	0.00	0.00	0.04	0.88	4.70	15.32	37.62	55.20	71.26
SIRO	0.10	0.30	0.86	2.64	7.62	17.96	34.04	46.56	58.12
AG1-PURE	0.06	0.30	0.56	1.88	5.40	13.54	26.86	37.58	48.76
AG1-10	0.08	0.38	0.62	1.96	5.70	13.54	26.02	38.02	48.04
AG1-20	0.18	0.60	0.98	2.50	7.22	15.90	28.58	41.74	52.20
AG1-30	0.30	0.66	1.36	3.12	8.16	16.96	31.74	42.74	53.36

Tabela 6.1: Porcentagem média de jobs atrasados

Técnica	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
EDD-FIRST	-	sim	sim	sim	sim	sim	sim	sim	-
EDD-ALL	-	sim	sim	sim	sim	sim	sim	sim	sim
SIRO	não	-	não	-	-	sim	sim	sim	sim
AG1-PURE	-	-	-	-	sim	sim	sim	sim	sim
AG1-10	-	-	-	-	sim	sim	sim	sim	sim
AG1-20	não	não	não	-	-	sim	sim	sim	sim
AG1-30	não	não	não	-	-	sim	sim	sim	sim

Tabela 6.2: Melhor que FIFO com 95% de confiança para a porcentagem média de jobs atrasados

detalhado da simulação mostrou que quando o valor de U aumenta, as *due dates* (prazos) dos jobs se tornam mais apertadas, e isso força todo novo job a esperar mais nas filas das máquinas. A estes jobs não é dado mais prazo já que o *allowance factor* é fixo para todos os testes. A situação chega a um ponto onde a regra EDD não consegue escolher um job que não esteja atrasado, pois ela escolhe os jobs com o prazo mais próximo do tempo atual do sistema, ou seja, aqueles que têm um menor tempo para terminarem definitivamente sua execução, e como estes jobs já passaram um tempo demasiado esperando nas filas, já estão potencialmente atrasados. Quando a regra SIRO é usada, os jobs são escalonados aleatoriamente, e então um job com prazo ainda suficiente tem a chance de ser escolhido e sair do sistema antes de sua *due date* ser ultrapassada.

Em geral, a tabela 6.1 mostra que as regras e o *guess and solve* com AGs mostram resultados parecidos quando as taxas de utilização são baixas, mas diferem significativamente quando U é mais alto. Isso ocorre pois, quando U é menor, menos jobs estão no sistema, e há pouco para ser escalonado, de forma que o comportamento dos jobs no sistema é parecido em todas as regras. Mas este comportamento muda gradativamente com o aumento dos valores de U, já que o número de jobs no sistema também aumenta e as técnicas passam a ter bem mais trabalho, podendo mostrar suas características. Sobre os resultados da tabela 6.2, nota-se que a partir de $U = 0.65$ é sempre mais vantajoso utilizar outra das técnicas de escalonamento mencionadas que não a FIFO.

Técnica	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
FIFO	0.28	2.34	8.89	13.19	18.20	24.66	34.68	47.33	56.96
EDD-FIRST	0.02	1.71	3.79	6.24	14.28	18.43	31.52	45.85	56.44
EDD-ALL	0.00	0.92	1.45	3.95	10.53	16.40	29.73	43.01	53.02
SIRO	0.00	3.83	14.52	23.30	31.59	46.92	53.79	64.49	74.24
AG1-PURE	0.42	3.60	5.98	9.43	13.65	21.60	31.44	45.33	56.18
AG1-10	2.74	9.66	5.39	9.13	15.47	20.33	32.15	49.90	62.22
AG1-20	1.30	6.87	7.58	14.38	17.43	21.67	34.78	52.69	64.47
AG1-30	2.82	10.88	14.64	15.99	20.97	27.46	38.17	53.55	68.10

Tabela 6.3: Porcentagem média de atraso

Técnica	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
EDD-FIRST	-	-	sim	sim	sim	sim	sim	sim	-
EDD-ALL	-	-	sim	sim	sim	sim	sim	sim	sim
SIRO	-	-	-	não	não	não	não	não	não
AG1-PURE	-	-	-	sim	sim	sim	sim	sim	sim
AG1-10	-	não	-	-	-	sim	sim	-	não
AG1-20	-	não	-	-	-	sim	-	não	não
AG1-30	não	não	-	-	-	-	não	não	não

Tabela 6.4: Melhor que FIFO com 95% de confiança para a porcentagem média de atraso

A figura 6.1 mostra a evolução da porcentagem média de jobs atrasados no sistema para os vários valores de U. Pode-se ver que o *guess and solve* constitui uma boa abordagem para este tipo de problema dinâmico.

6.2.2 Porcentagem Média de Atraso

Nesta seção, a tabela 6.3 mostra os resultados para a porcentagem média de atraso. A tabela 6.4 foi gerada da mesma forma e possui o mesmo propósito que a tabela 6.2.

Dessas duas tabelas, pode-se inferir que a regra EDD, principalmente em sua variante EDD-ALL, é o método mais eficiente para otimizar a porcentagem média de atraso. A regra EDD-ALL é apenas próxima da FIFO quando $U \leq 0.25$, sendo muito melhor que esta última nas outras taxas de utilização. Já a regra SIRO só se assemelha à FIFO até $U = 0.35$, e então se torna severamente pior. A regra EDD-FIRST está entre a FIFO e a EDD-ALL, mas mesmo assim pode ser considerada melhor que FIFO para $U > 0.25$ (com exceção de $U = 0.95$).

Os AGs mostram resultados não tão bons quanto os utilizados para otimizar a porcentagem de jobs atrasados, assemelhando-se mais à regra FIFO. Mesmo assim, o AG-PURE consegue ser estatisticamente melhor que FIFO para $U > 0.35$. Quando os erros do *guesser* são introduzidos, os resultados pioram, fazendo com que o AG-10, o AG-20 e o AG-30 tenham comportamentos ligeiramente oscilatórios, mas com a clara tendência de serem

Técnica	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
FIFO	68.03	71.85	78.84	85.86	103.93	123.88	174.98	228.53	291.72
SIRO	68.00	71.78	78.59	85.25	102.16	120.92	165.40	216.18	278.94
SPT-PURE	67.89	71.48	77.73	83.51	97.45	112.51	147.65	188.41	235.68
SPT-10	67.89	71.48	77.66	83.48	97.59	112.71	149.84	188.59	236.14
SPT-20	67.91	71.51	77.79	83.60	98.15	112.89	147.30	190.80	242.04
SPT-30	67.92	71.56	77.75	83.77	98.61	112.74	148.65	192.46	242.27
LSD-PURE	68.14	71.78	78.66	85.13	101.47	117.57	158.32	203.44	261.87
LSD-10	68.01	71.78	78.62	84.99	101.75	117.79	156.71	203.33	261.51
LSD-20	68.00	71.78	78.52	84.93	102.02	117.95	157.19	205.20	260.80
LSD-30	68.00	71.66	78.52	84.92	101.99	117.08	158.39	203.20	259.14
AG2-PURE	68.36	72.40	79.01	85.97	103.19	122.60	175.20	227.10	294.28
AG2-10	69.10	73.36	80.87	88.73	105.50	127.35	186.31	244.49	336.96
AG2-20	69.56	74.62	82.86	90.78	110.08	131.70	194.43	267.79	357.97
AG2-30	70.20	75.74	84.41	92.76	113.73	138.65	201.65	274.40	389.97

Tabela 6.5: Processing time médio

semelhantes a FIFO em utilizações mais baixas, e progressivamente piores que esta regra com valores mais altos de U. Uma configuração mais específica da implementação destes AGs pode baixar tais valores.

A figura 6.2 mostra uma representação gráfica da porcentagem média de atraso para todas as técnicas de escalonamento em cada taxa de utilização. Pode-se notar que a regra EDD mostra-se uma opção robusta quando se deve otimizar tanto a porcentagem de jobs atrasados quanto o atraso médio.

6.2.3 Processing Time Médio

Os resultados obtidos nas simulações para o *processing time* médio estão sumarizados na tabela 6.5. Como já mencionado, o *guess and solve* também é realizado com as regras SPT e LSD. A tabela 6.6 tem a mesma função das tabelas 6.2 e 6.4, sendo organizada da mesma forma.

Os números mostram a hegemonia da regra SPT na minimização do *processing time* médio, sendo esta a detentora dos melhores resultados para todos os valores de U. Tais melhores resultados se alternam entre SPT-PURE e SPT-10, o que indica, como na porcentagem de jobs atrasados, que um erro de 10% no *guess and solve* com SPT não se mostra ruim. O segundo lugar fica para SPT-20 e SPT-30, também em alternância, com ligeiro destaque para SPT-20. Mesmo em segundo lugar, essas duas últimas ficam próximas às duas primeiras. A regra LSD fica em terceiro, com resultados muito próximos entre LSD-PURE, LSD-10, LSD-20 e LSD-30. FIFO e SIRO andam próximas até que os valores de U atingem 0.65, a partir do qual elas se distanciam, e SIRO predomina.

Pode-se ver que, em se tratando de *processing time* médio, os erros do *guesser* fazem diferença pequena quando a fase de *solve* é realizada por uma *dispatching rule* como SPT

Técnica	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
SIRO	-	-	sim	sim	sim	sim	sim	sim	sim
SPT-PURE	sim	sim	sim	sim	sim	sim	sim	sim	sim
SPT-10	sim	sim	sim	sim	sim	sim	sim	sim	sim
SPT-20	sim	sim	sim	sim	sim	sim	sim	sim	sim
SPT-30	sim	sim	sim	sim	sim	sim	sim	sim	sim
LSD-PURE	-	-	-	sim	sim	sim	sim	sim	sim
LSD-10	-	-	-	sim	sim	sim	sim	sim	sim
LSD-20	-	-	sim	sim	sim	sim	sim	sim	sim
LSD-30	-	sim	sim	sim	sim	sim	sim	sim	sim
AG2-PURE	não	não	-	-	-	sim	-	-	-
AG2-10	não	não	não	não	não	não	não	não	não
AG2-20	não	não	não	não	não	não	não	não	não
AG2-30	não	não	não	não	não	não	não	não	não

Tabela 6.6: Melhor que FIFO com 95% de confiança para o processing time médio

ou LSD. Já para os algoritmos genéticos, é possível notar que os mesmos erros levam a resultados muito piores. O AG-PURE é o único que chega a ser melhor que FIFO numericamente, mas pela tabela 6.6 nota-se que ele é estatisticamente igual à FIFO na maioria dos casos. Os AG-10, AG-20 e AG-30 mostram-se sempre piores que FIFO em todas as taxas de utilização. Já o *guess and solve* com SPT e LSD é sempre melhor que FIFO em utilizações mais altas, ou pelo menos igual a ela nas mais baixas.

A figura 6.3 mostra graficamente os resultados das técnicas aplicadas a cada taxa de utilização. Por ela pode-ser ver a dominância das regras "do meio" (SPT e LSD) sobre as outras técnicas.

6.2.4 Tempo de execução dos AGs

Também foi analisado o custo computacional, em termos de tempo de processamento em segundos, dos algoritmos genéticos. Para estes testes, apenas casos onde $U = 0.95$ foram considerados, já que esta é a carga mais alta possível do sistema, e todas as outras taxas de utilização terão tempos de execução de seus AGs menores que o caso analisado. Foram utilizadas máquinas com processadores Intel Xeon duais e 1Gb de memória RAM. Como os AGs são executados a cada chegada de um dos 100 jobs, seus tempos de processamento evoluem em função da quantidade de jobs que chegou ao sistema. Três casos diferentes foram medidos: para o AG1 (otimização da porcentagem de jobs atrasados), para o AG2 (otimização da porcentagem média de atraso) e para o AG3 (otimização do *processing time* médio dos jobs).

Os tempos de processamento médio dos AGs são mostrados na figura 6.4. Pode-se notar que os tempos relativos ao AG1 e ao AG2 são muito parecidos, ficando seus valores mais altos em 3.16 e 3.02 segundos, respectivamente. Os tempos do AG3 já são bem mais altos, com um valor médio de pico de 14.88 segundos. Essa diferença deve-se

principalmente ao fato de o AG3 necessitar de muito mais gerações para otimizar sua função-objetivo, e portanto gastar mais tempo.

Analisando o pior caso entre todas as execuções dos AGs, o AG1 chegou a atingir 12.04 segundos. O AG2, por sua vez, atingiu 11.44 segundos. E o AG3 chegou aos 47.07 segundos. Mas estes casos são bem específicos. Além disso, sistemas de workflow geralmente operam com processos que duram tempos maiores, como dias ou mesmo meses. Portanto os tempos de execução dos AGs podem ser considerados adequados.

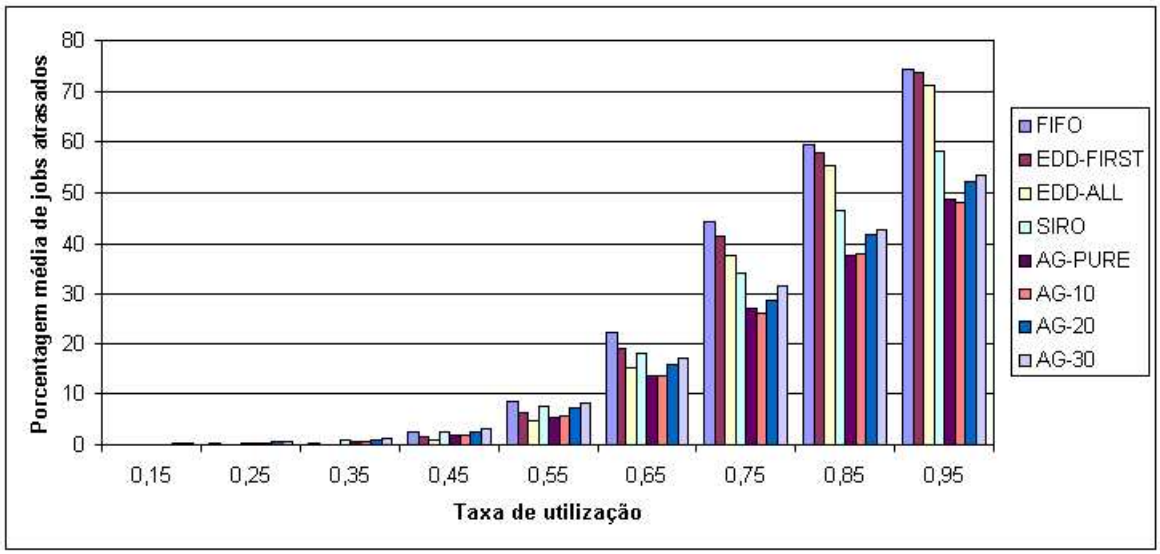


Figura 6.1: Porcentagem média de jobs atrasados

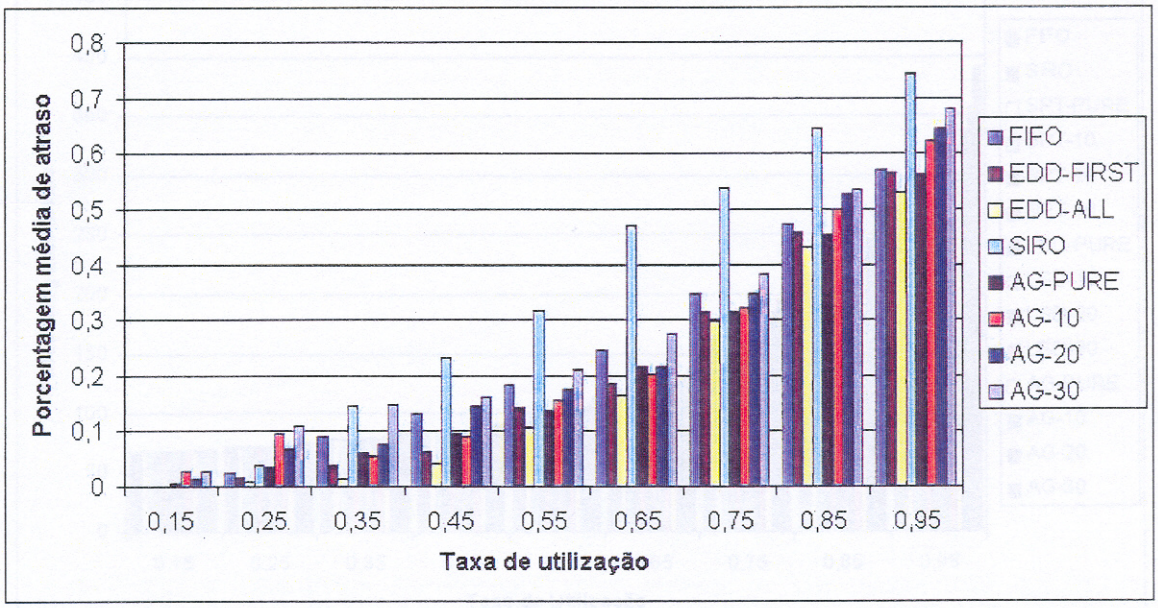


Figura 6.2: Porcentagem média de atraso

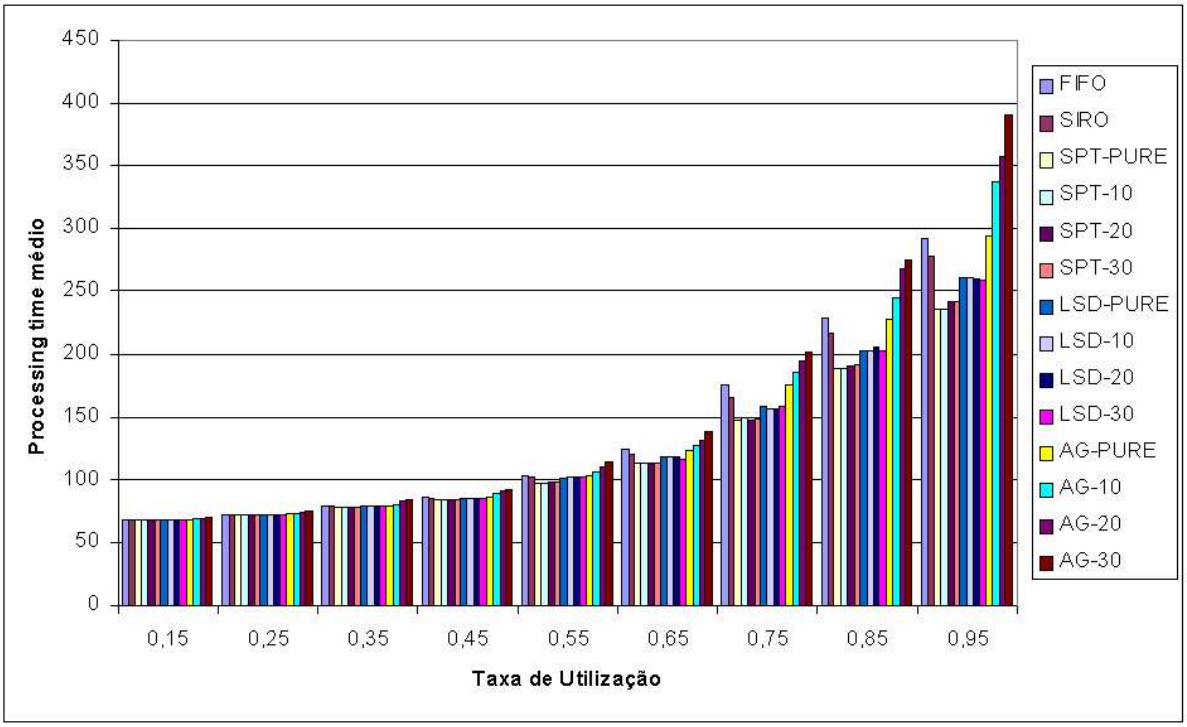


Figura 6.3: Processing time médio

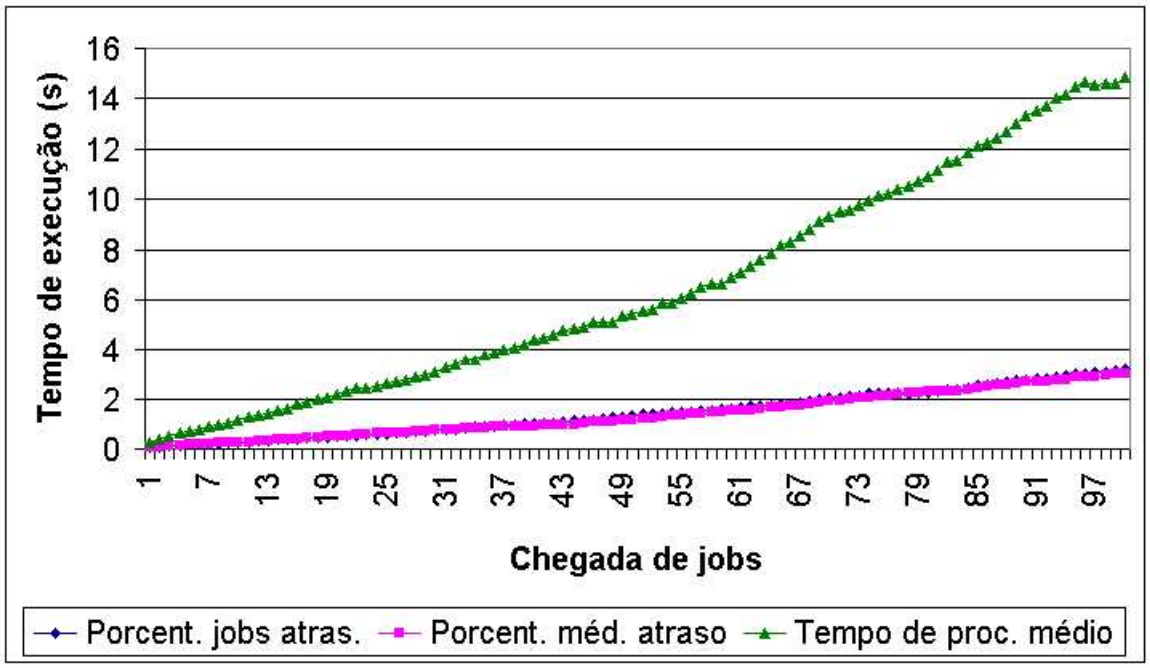


Figura 6.4: Tempo de execução dos AGs

Capítulo 7

Conclusões e Trabalhos Futuros

Sistemas de workflow atuam em diferentes cenários dentro das empresas dos dias atuais, e são um importante meio para se atingir maior eficiência com menor custo, condição fundamental para as companias que desejam manter-se vivas no mercado competitivo e globalizado que se configura hoje. Dentro desses sistemas, melhorias podem ser feitas quando aplicamos técnicas de escalonamento para ordenar o trabalho de forma a conseguir a otimização de quesitos valiosos, como o número de instâncias de processo atrasadas ou o tempo que cada instância passa no sistema.

Este trabalho apresenta uma discussão inicial sobre a aplicação de escalonamento a workflow, mostrando os problemas enfrentados e propondo um cenário de estudo onde incertezas quanto aos tempos de execução das instâncias em cada atividade, e quanto às rotas dessas instâncias dentro das definições de processo, estão presentes, e em um ambiente dinâmico, onde novas instâncias chegam enquanto outras estão executando.

Neste contexo, os resultados numéricos apontam para a superioridade de algumas técnicas de escalonamento sobre outras em cada situação estudada. Pode-se notar também que a metodologia aqui proposta de *guess and solve* mostra-se adequada para tratar problemas de escalonamento onde existem incertezas quanto ao tempo de execução dos jobs em cada máquina. Sendo o *guess and solve* um mapeamento da situação de cenários de workflow para problemas de escalonamento, mostra-se eficaz para servir de ponte entre workflows com incertezas de tempo e escalonamento, pelo menos quando a taxa de erro do *guesser* é limitada em 30%.

A análise estatística dos números mostra também que quase sempre é mais vantajoso utilizar uma outra forma de escalonamento de trabalho que não a política FIFO, a mais utilizada pelos sistemas de workflow, principalmente quando as taxas de utilização são iguais ou superiores a 65%, uma faixa onde se espera que a maioria das empresas se encontrem em termos de carga de trabalho.

Também se considera a discussão das dificuldades enfrentadas no mapeamento workflow-

escalonamento, do capítulo 4, como um importante resultado desta pesquisa, já que tal levantamento não parece encontrar par mesmo na literatura mais recente sobre workflow. Tal discussão também aponta para próximas dificuldades a serem enfrentadas em pesquisas como esta, principalmente quando se trata da avaliação do impacto que tais dificuldades têm em sistemas de workflow.

Um importante resultado foi a publicação de um artigo no 19^{th} *ACM Symposium on Applied Computig (SAC)*, e que pode ser verificado em [27]. Este artigo trata da discussão dos problemas apresentados no capítulo 4 deste trabalho, e contém um subconjunto dos resultados mostrados no capítulo 6, referentes à minimização da porcentagem de casos atrasados.

Por fim, apesar das novas dificuldades a serem enfrentadas, a utilização de técnicas de escalonamento mostra-se como um meio de fornecer uma potencial melhoria para os sistemas de workflow. Estes poderiam dispor de funcionalidades e meios que permitissem a aplicação dessas técnicas, por parte dos responsáveis pela administração do WFMS, ou mesmo de ferramentas capazes de auxiliar na escolha das técnicas mais adequadas para cada caso, além de ferramentas de monitoramento e administração avançada para permitir a análise de desempenho e o ajuste fino dessas funcionalidades.

7.1 Trabalhos Futuros

Ao final dos trabalhos para este Mestrado, os pesquisadores envolvidos tiveram a oportunidade de realizar uma análise crítica do que foi feito. Os pontos técnicos principais desta reflexão são os seguintes:

- O cenário estudado é na verdade uma estrutura de roteamento condicional (*OR-split*). Ou seja, algum conhecimento já foi construído sobre essa estrutura. Uma questão ainda em aberto é quão influentes são as atividades colocadas antes e depois das atividades diretamente envolvidas com o *OR-split*, com relação ao seu impacto nos resultados gerais.

- Os intervalos de tempo de processamento nas duas atividades constituintes do *OR-split* mencionado (atividades 2 e 3 da figura 5.1) são bem distintos entre si, justo para criar uma diferença entre o tempo que dois casos passam em cada uma dessas atividades. Mas a influência da escolha desses intervalos nos resultados não é conhecida.

- Os erros considerados para o *guess and solve* foram de 10%, 20% e 30%. Mas esses são valores limite para o erro do *guesser*, ou seja, o erro real de um *guesser* com "30% de erro" (para cada estimativa) está na verdade entre 0% e 30%. Isso causa

uma certa incerteza quanto ao real impacto do erro na fase de *guess* do *guess and solve*. A utilização de *guessers* com erros fixos pode esclarecer melhor esse ponto.

- Para o *OR-split* estudado, descobriu-se que regras como *Earliest Due Date* (EDD), que faz com que o caso com o prazo mais próximo de terminar seja executado primeiro, são melhores que a regra FIFO já mencionada. Além disso, o *guess and solve*, quando o erro é limitado a 30%, mostrou-se robusto e melhor que FIFO em cenários com cargas de trabalho mais altas, principalmente acima de 65%. Pergunta-se então se é possível, ao acumular conhecimentos também sobre as outras estruturas já mencionadas (roteamento seqüencial, paralelo e iterativo), desenvolver um método de composição destas em processos maiores, mantendo as propriedades de escalonamento já conhecidas

O conjunto desses pontos, acrescentado às dificuldades discutidas no capítulo 4, certamente fornece um amplo leque de oportunidades de trabalhos futuros nesta área de pesquisa.

Apêndice A

Abreviações Utilizadas

Esta seção apresenta uma lista das siglas e abreviações utilizadas durante este trabalho, para referência rápida.

AG. . . . . .Algoritmo genético
EDD. . . . . .Earliest Due Date
FIFO. . . . . .First In First Out
JSSP. . . . . .Job Shop Scheduling Problem
LSD. . . . . .Largest Successive Difference
OX. . . . . .Order-Based Crossover
PPX. . . . . .Precedence Preservative Crossover
RAM. . . . . .Random Access Memory
SGBD. . . . . .Sistema de Gerenciamento de Bancos de Dados
SIRO. . . . . .Service In Random Order
SPT. . . . . .Shortest Processing Time
UIMS. . . . . .User Interface Management System
WAPI. . . . . .Workflow Applications Programming Interface
WES. . . . . .Workflow Enactment Service
WfMC. . . . . .Workflow Management Coalition
WFMS. . . . . .Workflow Management System

Referências Bibliográficas

[1] Rob Allen. Workflow: An Introdution. In Layna Fischer, editor, *Workflow Handbook 2001*. Future Strategies in association with the Workflow Management Coalition, Lighthouse Point, FL, USA, 2001.

[2] Daniel Barbará, Sharad Mehrotra e Marek Rusinewics. Incas: Managing dynamic workflows in distributed environments. *Distributed Environments, Journal of Database Management*, 7(1), 1996.

[3] Christian Bierwirth, Herbert Kopfer, Dirk C. Mattfeld e Ivo Rixen. Genetic algorithm based scheduling in a dynamic manufacturing environment. In *Proc. of 1995 IEEE Conf. on Evolutionary Computation*, Piscataway, NJ, 1995. IEEE Press.

[4] Christian Bierwirth e Dirk C. Mattfeld. Minimizing job tardiness: Priority rules vs. adaptative scheduling. In *Ian C. Parmee (ed.), Adaptative Computing in Design and Manufacture*, London, 1998. Springer-Verlag.

[5] Christian Bierwirth e Dirk C. Mattfeld. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7:1–17, 1999.

[6] F. Casati, S. Ceri, B. Pernici e G. Pozzi. Conceptual modeling of workflows. In *Proceedings of the 14th International Object-Oriented and Entity-Relationship Modelling Conference*, volume 1021, pages 341–354. Springer-Verlag, 1995.

[7] Worfflow Management Coalition. Terminlogy and glossary. Technical report, Workflow Management Coalition, Whinchester, Hampshire - UK, 1999.

[8] Workflow Management Coalition. The workflow reference model (tc00-1003). Technical report, Workflow Management Colaition, Winchester, Hampshire - UK, 1995.

[9] Peter Cowling e Marcus Johansson. Using real time information for effective dynamic scheduling. *European Journal of Operational Research*, 139:230–244, 2002.

[10] Sofia Mara de Souza. *Estendendo Ambientes de Suporte a Trabalho Cooperativo com Base no Conceito de Workflow*. Dissertação de Mestrado. Instituto de Computação (IC) - Universidade Estadual de Campinas (UNICAMP), 2003.

[11] Nesime Tatbul et al. Workflow specification language and its scheduler. In *11th International Symposium on Computer and Information Systems*, Antalya, 1997.

[12] Paulo M. França, Alexandre Mendes e Pablo Moscato. A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132:224–242, 2001.

[13] R. Graham, E. Lawler, J. Lenstra e A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[14] Anant Singh Jain e Sheik Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

[15] Elizabeth. A. Kendall, M. T. Malkoun e C. H. Jiang. A methodology for developing agent based systems. In Chengqi Zhang e Dickson Lukose, editors, *First Australian Workshop on Distributed Artificial Intelligence*, Canberra, Australia, 1995.

[16] A. Kumar, W.M.P. van der Aalst e H.M.W. Verbeek. Dynamic work distribution in workflow management systems: How to balance quality and performance? *Journal of Management Information Systems*, 18(3):157–193, 2001.

[17] Manuel Laguna. Business process modeling, simulation and design. (forthcoming).

[18] Stephen R. Laurence e Edward C. Sewell. Heuristic, optimal, static and dynamic schedules when processing times are uncertain. *Journal of Operations Management*, 15:71–82, 1997.

[19] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[20] R. Gary Parker. *Deterministic Scheduling Theory*. Chapman & Hall, London, England, 1995.

[21] Michael Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[22] N. Raman e F. Brian Talbot. The job shop tardiness problem: A decomposition approach. *European Journal of Operational Research*, 69(2):187–199, 1993.

[23] Hajo A. Reijers. *Design and Control of Workflow Processes - Business Process Management for the Service Industry*. Springer-Verlag, Berlin, 2003.

[24] Leonardo Hartleben Reinehr. *Um Sistema de Gerenciamento de Workflows para Ambientes de Comunicação sem Fio*. Dissertação de Mestrado. Instituto de Computação (IC) - Universidade Estadual de Campinas (UNICAMP), 2002.

[25] Marek Rusinewics e Amit Sheth. Specification and execution of transactional workflows. In *Modern Database Systems: The Object Model, Interoperability and Beyond*, pages 592–620, 1995.

[26] Munindar P. Singh. Distributed scheduling of workflow computations. In *Proceedings of the Internation Conference on Data Engineering (ICDE)*, New Orleans, 1996.

[27] Gregório Baggio Tramontina, Jacques Wainer e Clarence Ellis. Applying scheduling techniques to minimize the number of late jobs in workflow systems. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 1396–1403, New York, NY, USA, 2004. ACM Press.

[28] Wil van der Aalst e Kees van Hee. *Workflow Management: Models, Methods and Systems*. The MIT Press - Massachussets Institute of Technology, 2002.

[29] J. Leon Zhao e Edward A. Stohr. Temporal workflow management in a claim handling system. In *Proceedings of the international joint conference on Work activities coordination and collaboration*, pages 187–195, New York, NY, USA, 1999. ACM Press.